

Docket No. 240490US2/hyc



IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

IN RE APPLICATION OF: Tsutomu OHISHI, et al.

GAU: 2622

SERIAL NO: 10/621,450

EXAMINER:

FILED: July 18, 2003

FOR: IMAGE FORMING APPARATUS, WRAPPING METHOD AND THE PROGRAM

REQUEST FOR PRIORITY

COMMISSIONER FOR PATENTS  
ALEXANDRIA, VIRGINIA 22313

SIR:

- ☐ Full benefit of the filing date of U.S. Application Serial Number , filed , is claimed pursuant to the provisions of 35 U.S.C. §120.
- ☐ Full benefit of the filing date(s) of U.S. Provisional Application(s) is claimed pursuant to the provisions of 35 U.S.C. §119(e): Application No. Date Filed
- ☒ Applicants claim any right to priority from any earlier filed applications to which they may be entitled pursuant to the provisions of 35 U.S.C. §119, as noted below.

In the matter of the above-identified application for patent, notice is hereby given that the applicants claim as priority:

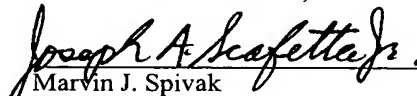
<u>COUNTRY</u>	<u>APPLICATION NUMBER</u>	<u>MONTH/DAY/YEAR</u>
JAPAN	2002-211685	July 19, 2002
JAPAN	2003-196228	July 14, 2003

Certified copies of the corresponding Convention Application(s)

- ☒ are submitted herewith
- ☐ will be submitted prior to payment of the Final Fee
- ☐ were filed in prior application Serial No. filed
- ☐ were submitted to the International Bureau in PCT Application Number  
Receipt of the certified copies by the International Bureau in a timely manner under PCT Rule 17.1(a) has been acknowledged as evidenced by the attached PCT/IB/304.
- ☐ (A) Application Serial No.(s) were filed in prior application Serial No. filed ; and
- ☐ (B) Application Serial No.(s)  
☐ are submitted herewith  
☐ will be submitted prior to payment of the Final Fee

Respectfully Submitted,

OBLON, SPIVAK, McCLELLAND,  
MAIER & NEUSTADT, P.C.

  
Marvin J. Spivak

Registration No. 24,913

Joseph A. Scafetta, Jr.  
Registration No. 26,803

Customer Number

22850

Tel. (703) 413-3000  
Fax. (703) 413-2220  
(OSMMN 05/03)

日 本 国 特 許 庁  
JAPAN PATENT OFFICE

別紙添付の書類に記載されている事項は下記の出願書類に記載されている事項と同一であることを証明する。

This is to certify that the annexed is a true copy of the following application as filed with this Office

出 願 年 月 日  
Date of Application:

2002年 7月19日

出 願 番 号  
Application Number:

特願2002-211685

[ ST.10/C ]:

[ JP 2002-211685 ]

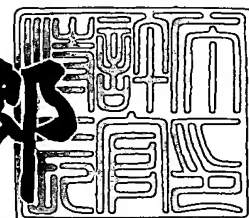
出 願 人  
Applicant(s):

株式会社リコー

2003年 5月 6日

特 許 庁 長 官  
Commissioner,  
Japan Patent Office

太田信一郎



出証番号 出証特2003-3032435

【書類名】 特許願

【整理番号】 0201298

【提出日】 平成14年 7月19日

【あて先】 特許庁長官殿

【国際特許番号】 G03G 21/00 370

【発明の名称】 画像形成装置およびラッピング処理方法並びにバージョン管理方法

【請求項の数】 33

【発明者】

    【住所又は居所】 東京都大田区中馬込 1 丁目 3 番 6 号 株式会社リコー内

    【氏名】 大石 勉

【発明者】

    【住所又は居所】 東京都大田区中馬込 1 丁目 3 番 6 号 株式会社リコー内

    【氏名】 秋吉 邦洋

【発明者】

    【住所又は居所】 東京都大田区中馬込 1 丁目 3 番 6 号 株式会社リコー内

    【氏名】 田中 浩行

【特許出願人】

    【識別番号】 000006747

    【氏名又は名称】 株式会社リコー

【代理人】

    【識別番号】 100089118

    【弁理士】

    【氏名又は名称】 酒井 宏明

【手数料の表示】

    【予納台帳番号】 036711

    【納付金額】 21,000円

【提出物件の目録】

    【物件名】 明細書 1

【物件名】 図面 1

【物件名】 要約書 1

【包括委任状番号】 9808514

【プルーフの要否】 要

【書類名】 明細書

【発明の名称】 画像形成装置およびラッピング処理方法並びにバージョン管理方法

【特許請求の範囲】

【請求項 1】 画像形成処理で使用されるハードウェア資源と、画像形成処理にかかるユーザサービスにそれぞれ固有の処理を行うアプリケーションのプロセスと、前記アプリケーションと前記ハードウェア資源との間に介在し、前記ユーザサービスを提供する際に、アプリケーションの少なくとも 2 つが共通的に必要とする前記ハードウェア資源の獲得要求、管理、実行制御並びに画像形成処理を行うコントロールサービスのプロセスと、前記コントロールサービスが予め定義された関数によって前記アプリケーションからの処理要求を受信可能とするアプリケーションプログラムインタフェースとを備えた画像形成装置であって、

前記コントロールサービスのいずれかのコンポーネントがバージョンアップしたときに、バージョン差の生じたアプリケーションからの関数呼び出しに対して整合性をとるラッピング処理手段

を備えたことを特徴とする画像形成装置。

【請求項 2】 画像形成処理で使用されるハードウェア資源と、画像形成処理にかかるユーザサービスにそれぞれ固有の処理を行うアプリケーションのプロセスと、前記アプリケーションと前記ハードウェア資源との間に介在し、前記ユーザサービスを提供する際に、アプリケーションの少なくとも 2 つが共通的に必要とする前記ハードウェア資源の獲得要求、管理、実行制御並びに画像形成処理を行うコントロールサービスのプロセスと、前記コントロールサービスが予め定義された関数によって前記アプリケーションからの処理要求を受信可能とするアプリケーションプログラムインタフェースとを備えた画像形成装置であって、

前記アプリケーションプログラムインタフェースがバージョンアップしたときに、バージョン差の生じたアプリケーションからの関数呼び出しに対して整合性をとるラッピング処理手段

を備えたことを特徴とする画像形成装置。

【請求項 3】 画像形成処理で使用されるハードウェア資源と、画像形成処

理にかかるユーザサービスにそれぞれ固有の処理を行うアプリケーションのプロセスと、前記アプリケーションと前記ハードウェア資源との間に介在し、前記ユーザサービスを提供する際に、アプリケーションの少なくとも2つが共通的に必要とする前記ハードウェア資源の獲得要求、管理、実行制御並びに画像形成処理を行うコントロールサービスのプロセスと、前記コントロールサービスが予め定義された関数によって前記アプリケーションからの処理要求を受信可能とするアプリケーションプログラムインタフェースとを備えた画像形成装置であって、

前記コントロールサービスから前記アプリケーションに対するメッセージを取捨選択するラッピング処理手段

を備えたことを特徴とする画像形成装置。

【請求項4】 前記ラッピング処理手段は、関数または関数の引数が追加されてバージョンアップしたときに、追加された関数または関数の引数に対応する機能が無視して整合性をとることを特徴とする請求項1または2に記載の画像形成装置。

【請求項5】 前記ラッピング処理手段は、関数または関数の引数が分割されてバージョンアップしたときに、分割された関数または関数の引数に応じてダミーの関数または関数の引数を補填して整合性をとることを特徴とする請求項1または2に記載の画像形成装置。

【請求項6】 前記ラッピング処理手段は、前記アプリケーションに対して通知しない特定のメッセージを予め設定可能にしたことを特徴とする請求項3に記載の画像形成装置。

【請求項7】 前記コントロールサービスをサーバとしたクライアントプロセスとして動作し、かつ前記アプリケーションをクライアントとしたサーバプロセスとして動作する仮想アプリケーションサービスをさらに備え、

前記ラッピング処理手段は、前記仮想アプリケーションサービスに含まれていることを特徴とする請求項1～6のいずれか一つに記載の画像形成装置。

【請求項8】 前記ラッピング処理手段は、前記仮想アプリケーションサービスのプロセス内部でスレッドとして生成されることを特徴とする請求項7に記載の画像形成装置。

【請求項 9】 前記アプリケーションの実行時に、共通の関数を利用して他のプログラムと結合させて実行形式のプログラムを作成する動的リンク手段をさらに備えていることを特徴とする請求項 1 ～ 8 のいずれか一つに記載の画像形成装置。

【請求項 1 0】 画像形成処理で使用されるハードウェア資源と、画像形成処理にかかるユーザサービスにそれぞれ固有の処理を行うアプリケーションのプロセスと、前記アプリケーションと前記ハードウェア資源との間に介在し、前記ユーザサービスを提供する際に、アプリケーションの少なくとも 2 つが共通的に必要とする前記ハードウェア資源の獲得要求、管理、実行制御並びに画像形成処理を行うコントロールサービスのプロセスと、前記コントロールサービスが予め定義された関数によって前記アプリケーションからの処理要求を受信可能とするアプリケーションプログラムインタフェースとを備えた画像形成装置であって、

前記アプリケーションと前記コントロールサービスのいずれかのコンポーネントとの間にバージョン差が生じたときに、そのバージョン差がサポート可能な範囲か否かを判定して、その判定結果を前記アプリケーションに通知するバージョン管理手段

を備えたことを特徴とする画像形成装置。

【請求項 1 1】 画像形成処理で使用されるハードウェア資源と、画像形成処理にかかるユーザサービスにそれぞれ固有の処理を行うアプリケーションのプロセスと、前記アプリケーションと前記ハードウェア資源との間に介在し、前記ユーザサービスを提供する際に、アプリケーションの少なくとも 2 つが共通的に必要とする前記ハードウェア資源の獲得要求、管理、実行制御並びに画像形成処理を行うコントロールサービスのプロセスと、前記コントロールサービスが予め定義された関数によって前記アプリケーションからの処理要求を受信可能とするアプリケーションプログラムインタフェースとを備えた画像形成装置であって、

前記アプリケーションと前記アプリケーションプログラムインタフェースとの間にバージョン差が生じたときに、そのバージョン差がサポート可能な範囲か否かを判定して、その判定結果を前記アプリケーションに通知するバージョン管理手段

を備えたことを特徴とする画像形成装置。

【請求項 1 2】 前記バージョン管理手段は、サポート可能な範囲のバージョンか否かを前記アプリケーションのバージョン番号に基づいて判定することを特徴とする請求項 1 0 または 1 1 に記載の画像形成装置。

【請求項 1 3】 前記バージョン管理手段は、サポート可能な範囲のバージョン番号が登録されたバージョン管理テーブルをさらに備え、前記アプリケーションのバージョン番号を前記バージョン管理テーブルと比較して判定することを特徴とする請求項 1 2 に記載の画像形成装置。

【請求項 1 4】 画像形成処理で使用されるハードウェア資源と、画像形成処理にかかるユーザサービスにそれぞれ固有の処理を行うアプリケーションのプロセスと、前記アプリケーションと前記ハードウェア資源との間に介在し、前記ユーザサービスを提供する際に、アプリケーションの少なくとも 2 つが共通的に必要とする前記ハードウェア資源の獲得要求、管理、実行制御並びに画像形成処理を行うコントロールサービスのプロセスと、前記コントロールサービスが予め定義された関数によって前記アプリケーションからの処理要求を受信可能とするアプリケーションプログラムインタフェースとを備えた画像形成装置であって、

前記アプリケーションが使用する関数を全種類抽出し、その抽出した各関数と、前記コントロールサービスのいずれかのコンポーネントとの間にバージョン差が生じたときに、そのバージョン差がサポート可能な範囲か否かを判定して、その判定結果を前記アプリケーションに通知するバージョン管理手段

を備えたことを特徴とする画像形成装置。

【請求項 1 5】 画像形成処理で使用されるハードウェア資源と、画像形成処理にかかるユーザサービスにそれぞれ固有の処理を行うアプリケーションのプロセスと、前記アプリケーションと前記ハードウェア資源との間に介在し、前記ユーザサービスを提供する際に、アプリケーションの少なくとも 2 つが共通的に必要とする前記ハードウェア資源の獲得要求、管理、実行制御並びに画像形成処理を行うコントロールサービスのプロセスと、前記コントロールサービスが予め定義された関数によって前記アプリケーションからの処理要求を受信可能とするアプリケーションプログラムインタフェースとを備えた画像形成装置であって、



前記アプリケーションが使用する関数を全種類抽出し、その抽出した各関数と、前記アプリケーションプログラムインタフェースとの間にバージョン差が生じたときに、そのバージョン差がサポート可能な範囲か否かを判定して、その判定結果を前記アプリケーションに通知するバージョン管理手段を備えたことを特徴とする画像形成装置。

【請求項 1 6】 前記バージョン管理手段は、サポート可能な範囲のバージョンか否かを前記アプリケーションが使用する全ての関数番号とバージョン番号とに基づいて判定することを特徴とする請求項 1 4 または 1 5 に記載の画像形成装置。

【請求項 1 7】 前記バージョン管理手段は、サポート可能な範囲の全ての関数番号とバージョン番号とが登録されたバージョン管理テーブルをさらに備え、前記アプリケーションが使用する全ての関数番号とバージョン番号を前記バージョン管理テーブルと比較して判定することを特徴とする請求項 1 6 に記載の画像形成装置。

【請求項 1 8】 前記コントロールサービスをサーバとしたクライアントプロセスとして動作し、かつ前記アプリケーションをクライアントとしたサーバプロセスとして動作する仮想アプリケーションサービスをさらに備え、

前記バージョン管理手段は、前記仮想アプリケーションサービスに含まれていることを特徴とする請求項 1 0 ～ 1 7 のいずれか一つに記載の画像形成装置。

【請求項 1 9】 前記バージョン管理手段は、前記仮想アプリケーションサービスのプロセス内部でスレッドとして生成されることを特徴とする請求項 1 8 に記載の画像形成装置。

【請求項 2 0】 画像形成処理で使用されるハードウェア資源と、画像形成処理にかかるユーザサービスにそれぞれ固有の処理を行うアプリケーションのプロセスと、前記アプリケーションと前記ハードウェア資源との間に介在し、前記ユーザサービスを提供する際に、アプリケーションの少なくとも 2 つが共通的に必要とする前記ハードウェア資源の獲得要求、管理、実行制御並びに画像形成処理を行うコントロールサービスのプロセスと、前記コントロールサービスが予め定義された関数によって前記アプリケーションからの処理要求を受信可能とする

アプリケーションプログラムインタフェースとを備えた画像形成装置上で、前記アプリケーションとのバージョン差を吸収するラッピング処理方法であって、

前記コントロールサービスのいずれかのコンポーネントがバージョンアップしたときに、バージョン差の生じたアプリケーションからの関数呼び出しに対して整合性をとるラッピング処理ステップ

を含むことを特徴とするラッピング処理方法。

【請求項 2 1】 画像形成処理で使用されるハードウェア資源と、画像形成処理にかかるユーザサービスにそれぞれ固有の処理を行うアプリケーションのプロセスと、前記アプリケーションと前記ハードウェア資源との間に介在し、前記ユーザサービスを提供する際に、アプリケーションの少なくとも 2 つが共通的に必要とする前記ハードウェア資源の獲得要求、管理、実行制御並びに画像形成処理を行うコントロールサービスのプロセスと、前記コントロールサービスが予め定義された関数によって前記アプリケーションからの処理要求を受信可能とするアプリケーションプログラムインタフェースとを備えた画像形成装置上で、前記アプリケーションとのバージョン差を吸収するラッピング処理方法であって、

前記アプリケーションプログラムインタフェースがバージョンアップしたときに、バージョン差の生じたアプリケーションからの関数呼び出しに対して整合性をとるラッピング処理ステップ

を含むことを特徴とするラッピング処理方法。

【請求項 2 2】 画像形成処理で使用されるハードウェア資源と、画像形成処理にかかるユーザサービスにそれぞれ固有の処理を行うアプリケーションのプロセスと、前記アプリケーションと前記ハードウェア資源との間に介在し、前記ユーザサービスを提供する際に、アプリケーションの少なくとも 2 つが共通的に必要とする前記ハードウェア資源の獲得要求、管理、実行制御並びに画像形成処理を行うコントロールサービスのプロセスと、前記コントロールサービスが予め定義された関数によって前記アプリケーションからの処理要求を受信可能とするアプリケーションプログラムインタフェースとを備えた画像形成装置上で、前記アプリケーションとのバージョン差を吸収するラッピング処理方法であって、

前記コントロールサービスから前記アプリケーションに対するメッセージを取

## 捨選択するラッピング処理ステップ

を含むことを特徴とするラッピング処理方法。

【請求項 2 3】 前記ラッピング処理ステップは、関数または関数の引数が追加されてバージョンアップしたときに、追加された関数または関数の引数に対応する機能を見捨てして整合性をとることを特徴とする請求項 2 0 または 2 1 に記載のラッピング処理方法。

【請求項 2 4】 前記ラッピング処理ステップは、関数または関数の引数が分割されてバージョンアップしたときに、分割された関数または関数の引数に応じてダミーの関数または関数の引数を補填して整合性をとることを特徴とする請求項 2 0 または 2 1 に記載のラッピング処理方法。

【請求項 2 5】 前記ラッピング処理ステップは、前記アプリケーションに対して通知しない特定のメッセージを予め設定可能にしたことを特徴とする請求項 2 2 に記載のラッピング処理方法。

【請求項 2 6】 画像形成処理で使用するハードウェア資源と、画像形成処理にかかるユーザサービスにそれぞれ固有の処理を行うアプリケーションのプロセスと、前記アプリケーションと前記ハードウェア資源との間に介在し、前記ユーザサービスを提供する際に、アプリケーションの少なくとも 2 つが共通的に必要とする前記ハードウェア資源の獲得要求、管理、実行制御並びに画像形成処理を行うコントロールサービスのプロセスと、前記コントロールサービスが予め定義された関数により前記アプリケーションからの処理要求を受信可能とするアプリケーションプログラムインタフェースとを備えた画像形成装置上で、前記アプリケーションとの間のバージョンを管理するバージョン管理方法であって、

前記アプリケーションと前記コントロールサービスのいずれかのコンポーネントとの間にバージョン差が生じたときに、そのバージョン差がサポート可能な範囲か否かを判定して、その判定結果を前記アプリケーションに通知するバージョン管理ステップ

を含むことを特徴とするバージョン管理方法。

【請求項 2 7】 画像形成処理で使用するハードウェア資源と、画像形成処理にかかるユーザサービスにそれぞれ固有の処理を行うアプリケーションのプ

ロセスと、前記アプリケーションと前記ハードウェア資源との間に介在し、前記ユーザサービスを提供する際に、アプリケーションの少なくとも2つが共通的に必要とする前記ハードウェア資源の獲得要求、管理、実行制御並びに画像形成処理を行うコントロールサービスのプロセスと、前記コントロールサービスが予め定義された関数により前記アプリケーションからの処理要求を受信可能とするアプリケーションプログラムインタフェースとを備えた画像形成装置上で、前記アプリケーションとの間のバージョンを管理するバージョン管理方法であって、

前記アプリケーションと前記アプリケーションプログラムインタフェースとの間にバージョン差が生じたときに、そのバージョン差がサポート可能な範囲か否かを判定して、その判定結果を前記アプリケーションに通知するバージョン管理ステップ

を含むことを特徴とするバージョン管理方法。

【請求項28】 前記バージョン管理ステップは、サポート可能な範囲のバージョン差か否かを前記アプリケーションのバージョン番号に基づいて判定することを特徴とする請求項26または27に記載のバージョン管理方法。

【請求項29】 前記バージョン管理ステップは、サポート可能な範囲のバージョン番号が登録されたバージョン管理テーブルと比較して判定することを特徴とする請求項28に記載のバージョン管理方法。

【請求項30】 画像形成処理で使用されるハードウェア資源と、画像形成処理にかかるユーザサービスにそれぞれ固有の処理を行うアプリケーションのプロセスと、前記アプリケーションと前記ハードウェア資源との間に介在し、前記ユーザサービスを提供する際に、アプリケーションの少なくとも2つが共通的に必要とする前記ハードウェア資源の獲得要求、管理、実行制御並びに画像形成処理を行うコントロールサービスのプロセスと、前記コントロールサービスが予め定義された関数により前記アプリケーションからの処理要求を受信可能とするアプリケーションプログラムインタフェースとを備えた画像形成装置上で、前記アプリケーションが使用する全関数についてサポートが可能か否かを判定するバージョン管理方法であって、

前記アプリケーションが使用する関数を全種類抽出し、その抽出した各関数と

、前記コントロールサービスのいずれかのコンポーネントとの間にバージョン差が生じたときに、そのバージョン差がサポート可能な範囲か否かを判定して、その判定結果を前記アプリケーションに通知するバージョン管理ステップを含むことを特徴とするバージョン管理方法。

【請求項 3 1】 画像形成処理で使用されるハードウェア資源と、画像形成処理にかかるユーザサービスにそれぞれ固有の処理を行うアプリケーションのプロセスと、前記アプリケーションと前記ハードウェア資源との間に介在し、前記ユーザサービスを提供する際に、アプリケーションの少なくとも 2 つが共通的に必要とする前記ハードウェア資源の獲得要求、管理、実行制御並びに画像形成処理を行うコントロールサービスのプロセスと、前記コントロールサービスが予め定義された関数により前記アプリケーションからの処理要求を受信可能とするアプリケーションプログラムインタフェースとを備えた画像形成装置上で、前記アプリケーションが使用する全関数についてサポートが可能か否かを判定するバージョン管理方法であって、

前記アプリケーションが使用する関数を全種類抽出し、その抽出した各関数と、前記アプリケーションプログラムインタフェースとの間にバージョン差が生じたときに、そのバージョン差がサポート可能な範囲か否かを判定して、その判定結果を前記アプリケーションに通知するバージョン管理ステップ

を含むことを特徴とするバージョン管理方法。

【請求項 3 2】 前記バージョン管理ステップは、サポート可能な範囲のバージョン差か否かを前記アプリケーションが使用する全ての関数番号とバージョン番号とに基づいて判定することを特徴とする請求項 3 0 または 3 1 に記載のバージョン管理方法。

【請求項 3 3】 前記バージョン管理ステップは、前記アプリケーションが使用する全ての関数番号とバージョン番号を、サポート可能な範囲の関数番号とバージョン番号が登録されたバージョン管理テーブルと比較して判定することを特徴とする請求項 3 2 に記載のバージョン管理方法。

【発明の詳細な説明】

【0 0 0 1】

## 【発明の属する技術分野】

この発明は、コピー、プリンタ、スキャナおよびファクシミリなどの画像形成処理にかかるユーザサービスを提供する画像形成装置に搭載されるアプリケーションとの間でバージョン差が生じた場合に、バージョン不整合による不具合発生を防止することができる画像形成装置およびラッピング処理方法並びにバージョン管理方法に関するものである。

## 【0002】

## 【従来の技術】

近年では、プリンタ、コピー、ファクシミリ、スキャナなどの各装置の機能を1つの筐体内に収納した画像形成装置（以下、「複合機」という。）が一般的に知られている。この複合機は、1つの筐体内に表示部、印刷部および撮像部などを設けるとともに、プリンタ、コピーおよびファクシミリ装置にそれぞれ対応する3種類のソフトウェアを設け、これらのソフトウェアを切り替えることによって、当該装置をプリンタ、コピー、スキャナまたはファクシミリ装置として動作させるものである。

## 【0003】

このような従来の複合機では、プリンタ、コピー、ファクシミリ、スキャナなどの各機能単位でアプリケーションプログラムが起動され、ハードウェア資源にアクセスする機能を持ち合わせている。その際、アプリケーションプログラムとオペレーティングシステム（OS）のバージョンが同じことが前提となるが、例えば、OSをバージョンアップしてアプリケーションとの間でバージョン差が生じた場合、今まで使えていた機能が使えなくなったり、アプリケーションそのものが起動しなくなったりすることがある。

## 【0004】

このため、従来の複合機では、OSなどをバージョンアップすると、それに伴ってアプリケーションの方もコンパイルし直すことで、常に対応したバージョン関係にあることが要請されている。

## 【0005】

## 【発明が解決しようとする課題】

ところで、このような従来の複合機では、プリンタ、コピー、スキャナおよびファクシミリ装置に対応するソフトウェアがそれぞれ別個に設けられているため、各ソフトウェアの開発に多大の時間を要する。このため、出願人は、表示部、印刷部および撮像部などの画像形成処理で使用するハードウェア資源を有し、プリンタ、コピーまたはファクシミリなどの各ユーザサービスにそれぞれ固有の処理を行うアプリケーションを複数搭載し、これらのアプリケーションとハードウェア資源との間に介在して、ユーザサービスを提供する際に、アプリケーションの少なくとも2つが共通的に必要とするハードウェア資源の管理、実行制御並びに画像形成処理を行う各種コントロールサービスからなるプラットフォームを備えた画像形成装置（複合機）を発明した。

## 【 0 0 0 6 】

このような新規な複合機では、アプリケーションとコントロールサービスとを別個に設けているため、複合機の出荷後にユーザもしくは第三者であるサードベンダーが新規なアプリケーションを開発して複合機に搭載することが可能となり、これによって多種多様な機能を提供することが可能となる。

## 【 0 0 0 7 】

このような新規な複合機では、アプリケーションの少なくとも2つが共通的に必要とするサービスを提供するコントロールサービスをアプリケーションと別個に設けた構成となっているため、新規アプリケーションを開発する場合には、各種コントロールサービスとのプロセス間通信を実現する処理をソースコードで記述する必要がある。しかしながら、新規アプリケーションを開発する場合、各コントロールサービスが提供する関数やメッセージなどを正確に把握した上で、予め規定された手順で記述しなければならず、各コントロールサービスやアプリケーションからの処理要求を受信可能とするアプリケーションプログラムインタフェース（API）をデバッグや機能追加によってバージョンアップが繰り返されると、ベンダーはどのバージョンに合わせてアプリケーション開発すればよいかが非常に分かり難くなり、アプリケーションの開発自体を阻害するおそれがある。このことは、固定的な複数の機能を寄せ集めた従来の複合機では問題にならなかった新規な課題である。

## 【 0 0 0 8 】

また、各コントロールサービスとアプリケーションのインタフェース（API）のすべてを、新規アプリケーションを開発するサードベンダなどの第三者に開示することは、プログラムの秘匿性の面から言って好ましくない。特に、複合機のシステムに大きく影響を与えるような箇所については、第三者に対して隠蔽することで直接コントロールサービスに対してアクセスされるのを回避することが可能となり、複合機のセキュリティ面および障害発生を未然に防止する面からも好ましい。

## 【 0 0 0 9 】

この発明は上記に鑑みてなされたもので、多数のプロセスが実行される画像形成装置上で動作するアプリケーションとコントロールサービスおよびインタフェースとの間でバージョン管理を行い、バージョン差が生じた場合でもバージョン差を吸収して整合性をとるようにすると共に、アプリケーションとコントロールサービス間の複雑なインタフェースを隠蔽して重要なインタフェースの秘匿性を確保すると共に、特定のインタフェースを開示することによって新規アプリケーションの開発効率を向上させることができる画像形成装置およびラッピング処理方法並びにバージョン管理方法を得ることを目的とする。

## 【 0 0 1 0 】

## 【課題を解決するための手段】

上記目的を達成するため、請求項1にかかる発明は、画像形成処理で使用されるハードウェア資源と、画像形成処理にかかるユーザサービスにそれぞれ固有の処理を行うアプリケーションのプロセスと、前記アプリケーションと前記ハードウェア資源との間に介在し、前記ユーザサービスを提供する際に、アプリケーションの少なくとも2つが共通的に必要とする前記ハードウェア資源の獲得要求、管理、実行制御並びに画像形成処理を行うコントロールサービスのプロセスと、前記コントロールサービスが予め定義された関数によって前記アプリケーションからの処理要求を受信可能とするアプリケーションプログラムインタフェースとを備えた画像形成装置であって、前記コントロールサービスのいずれかのコンポーネントがバージョンアップしたときに、バージョン差の生じたアプリケーション



ンからの関数呼び出しに対して整合性をとるラッピング処理手段を備えたことを特徴とする。

## 【0011】

この請求項1にかかる発明によれば、コントロールサービスのいずれかのコンポーネントがバージョンアップしたとき、ラッピング処理手段によってバージョン差の生じたアプリケーションからの関数呼び出しに対して整合性をとるようにしたため、バージョン差が吸収され、バージョン不整合による不具合発生を防止することができる。

## 【0012】

また、請求項2にかかる発明は、画像形成処理で使用されるハードウェア資源と、画像形成処理にかかるユーザサービスにそれぞれ固有の処理を行うアプリケーションのプロセスと、前記アプリケーションと前記ハードウェア資源との間に介在し、前記ユーザサービスを提供する際に、アプリケーションの少なくとも2つが共通的に必要とする前記ハードウェア資源の獲得要求、管理、実行制御並びに画像形成処理を行うコントロールサービスのプロセスと、前記コントロールサービスが予め定義された関数によって前記アプリケーションからの処理要求を受信可能とするアプリケーションプログラムインタフェースとを備えた画像形成装置であって、前記アプリケーションプログラムインタフェースがバージョンアップしたときに、バージョン差の生じたアプリケーションからの関数呼び出しに対して整合性をとるラッピング処理手段を備えたことを特徴とする。

## 【0013】

この請求項2にかかる発明によれば、アプリケーションプログラムインタフェースがバージョンアップしたとき、ラッピング処理手段によってバージョン差の生じたアプリケーションからの関数呼び出しに対して整合性をとるようにしたため、バージョン差が吸収され、バージョン不整合による不具合発生を防止することができる。

## 【0014】

また、請求項3にかかる発明は、画像形成処理で使用されるハードウェア資源と、画像形成処理にかかるユーザサービスにそれぞれ固有の処理を行うアプリケ

ーションのプロセスと、前記アプリケーションと前記ハードウェア資源との間に介在し、前記ユーザサービスを提供する際に、アプリケーションの少なくとも2つが共通的に必要とする前記ハードウェア資源の獲得要求、管理、実行制御並びに画像形成処理を行うコントロールサービスのプロセスと、前記コントロールサービスが予め定義された関数によって前記アプリケーションからの処理要求を受信可能とするアプリケーションプログラムインタフェースとを備えた画像形成装置であって、前記コントロールサービスから前記アプリケーションに対するメッセージを取捨選択するラッピング処理手段を備えたことを特徴とする。

## 【0015】

この請求項3にかかる発明によれば、ラッピング処理手段によってコントロールサービスからアプリケーションに対するメッセージを取捨選択するため、サードベンダなどの第三者に対してはコントロールサービスとの間のインタフェースを隠蔽し、重要なインタフェースの秘匿性を確保して、第三者からコントロールサービスに直接アクセスされるのを回避することにより、画像形成装置におけるセキュリティ面が確保され、障害発生を未然に防止することができる。

## 【0016】

また、請求項4にかかる発明は、請求項1または2に記載の画像形成装置において、前記ラッピング処理手段は、関数または関数の引数が追加されてバージョンアップしたときに、追加された関数または関数の引数に対応する機能は無視して整合性をとることを特徴とする。

## 【0017】

この請求項4にかかる発明によれば、関数または関数の引数が追加されてバージョンアップした場合、ラッピング処理手段はアプリケーションに対して提供する関数または関数の引数を変更しなければ、追加された機能は使えないが既存の機能はそのまま使えるため、追加機能は無視すればアプリケーションを再コンパイルすることなくバージョン差を吸収することができる。

## 【0018】

また、請求項5にかかる発明は、請求項1または2に記載の画像形成装置において、前記ラッピング処理手段は、関数または関数の引数が分割されてバージョ

ンアップしたときに、分割された関数または関数の引数に応じてダミーの関数または関数の引数を補填して整合性をとることを特徴とする。

## 【 0 0 1 9 】

この請求項 5 にかかる発明によれば、関数または関数の引数が分割されてバージョンアップした場合、対応する関数や関数の引数が増えるため、そのままでは関数呼び出しを行うことができず、ラッピング処理手段によって増加した関数や関数の引数に対応するダミーの関数または関数の引数を補填することによって整合性をとることができ、バージョン差を吸収することができる。

## 【 0 0 2 0 】

また、請求項 6 にかかる発明は、請求項 3 に記載の画像形成装置において、前記ラッピング処理手段は、前記アプリケーションに対して通知しない特定のメッセージを予め設定可能にしたことを特徴とする。

## 【 0 0 2 1 】

この請求項 6 にかかる発明によれば、特定のメッセージをアプリケーションに対して通知しないように、ラッピング処理手段に対して予め設定することが可能なため、重要なインタフェースの秘匿性を確保することができる。

## 【 0 0 2 2 】

また、請求項 7 にかかる発明は、請求項 1 ～ 6 のいずれか一つに記載の画像形成装置において、前記コントロールサービスをサーバとしたクライアントプロセスとして動作し、かつ前記アプリケーションをクライアントとしたサーバプロセスとして動作する仮想アプリケーションサービスをさらに備え、前記ラッピング処理手段は、前記仮想アプリケーションサービスに含まれていることを特徴とする。

## 【 0 0 2 3 】

この請求項 7 にかかる発明によれば、ラッピング処理手段は仮想アプリケーションサービスに含まれており、この仮想アプリケーションサービスは、コントロールサービスをサーバとしたクライアントプロセスとして動作し、かつアプリケーションをクライアントとしたサーバプロセスとして動作するため、バージョン差が生じた場合にバージョン差を吸収して整合性をとることができ、また、コン

トロールサービスからアプリケーションに対するメッセージを選択することによって、インタフェースを隠蔽して重要なインタフェースの秘匿性を確保しつつ、新規アプリケーションのプログラム開発を効率的に行うことができる。

## 【 0 0 2 4 】

また、請求項 8 にかかる発明は、請求項 7 に記載の画像形成装置において、前記ラッピング処理手段は、前記仮想アプリケーションサービスのプロセス内部でスレッドとして生成されることを特徴とする。

## 【 0 0 2 5 】

この請求項 8 にかかる発明によれば、ラッピング処理手段は、仮想アプリケーションサービスのプロセス内部にスレッドとして生成され、実行されるため、仮想アプリケーションサービスのプロセス内部で軽快に動作させることができると共に、プロセス内部に複数個のスレッドを生成した場合であっても、それらを並行処理することができる。

## 【 0 0 2 6 】

また、請求項 9 にかかる発明は、請求項 1 ～ 8 のいずれか一つに記載の画像形成装置において、前記アプリケーションの実行時に、共通の関数を利用して他のプログラムと結合させて実行形式のプログラムを作成する動的リンク手段をさらに備えていることを特徴とする。

## 【 0 0 2 7 】

この請求項 9 にかかる発明によれば、アプリケーションの実行時に、共通の関数を利用して他のプログラムと結合させることにより実行形式のプログラムを作成する動的リンク手段を備えているため、アプリケーションとコントロールサービスあるいはインタフェース間のバージョン差の吸収力が増大し、整合性が一層とれるようになると共に、インタフェースを隠蔽して重要なインタフェースの秘匿性をより強固に確保することができる。

## 【 0 0 2 8 】

また、請求項 1 0 にかかる発明は、画像形成処理で使用するハードウェア資源と、画像形成処理にかかるユーザサービスにそれぞれ固有の処理を行うアプリケーションのプロセスと、前記アプリケーションと前記ハードウェア資源との間

に介在し、前記ユーザサービスを提供する際に、アプリケーションの少なくとも2つが共通的に必要とする前記ハードウェア資源の獲得要求、管理、実行制御並びに画像形成処理を行うコントロールサービスのプロセスと、前記コントロールサービスが予め定義された関数によって前記アプリケーションからの処理要求を受信可能とするアプリケーションプログラムインタフェースとを備えた画像形成装置であって、前記アプリケーションと前記コントロールサービスのいずれかのコンポーネントとの間にバージョン差が生じたときに、そのバージョン差がサポート可能な範囲か否かを判定して、その判定結果を前記アプリケーションに通知するバージョン管理手段を備えたことを特徴とする。

## 【 0 0 2 9 】

この請求項10にかかる発明によれば、アプリケーションとコントロールサービスのいずれかのコンポーネントとの間でバージョン差を生じた場合に、そのバージョン差がサポート可能な範囲か否かを判定し、その判定結果をアプリケーションに通知するようにしたため、例えばアプリケーションの登録時などにバージョンの整合性を判定すれば、アプリケーションの実行前にバージョンの適合性の有無が判断できることから、障害の発生を未然に防止することができる。

## 【 0 0 3 0 】

また、請求項11にかかる発明は、画像形成処理で使用されるハードウェア資源と、画像形成処理にかかるユーザサービスにそれぞれ固有の処理を行うアプリケーションのプロセスと、前記アプリケーションと前記ハードウェア資源との間に介在し、前記ユーザサービスを提供する際に、アプリケーションの少なくとも2つが共通的に必要とする前記ハードウェア資源の獲得要求、管理、実行制御並びに画像形成処理を行うコントロールサービスのプロセスと、前記コントロールサービスが予め定義された関数によって前記アプリケーションからの処理要求を受信可能とするアプリケーションプログラムインタフェースとを備えた画像形成装置であって、前記アプリケーションと前記アプリケーションプログラムインタフェースとの間にバージョン差が生じたときに、そのバージョン差がサポート可能な範囲か否かを判定して、その判定結果を前記アプリケーションに通知するバージョン管理手段を備えたことを特徴とする。

## 【 0 0 3 1 】

この請求項 1 1 にかかる発明によれば、アプリケーションとアプリケーションプログラムインタフェースとの間でバージョン差が生じた場合に、そのバージョン差がサポート可能な範囲か否かを判定し、その判定結果をアプリケーションに通知するようにしたため、例えばアプリケーションの登録時などにバージョンの整合性を判定すれば、アプリケーションの実行前にバージョンの適合性の有無が判断できることから、障害の発生を未然に防止することができる。

## 【 0 0 3 2 】

また、請求項 1 2 にかかる発明は、請求項 1 0 または 1 1 に記載の画像形成装置において、前記バージョン管理手段は、サポート可能な範囲のバージョンか否かを前記アプリケーションのバージョン番号に基づいて判定することを特徴とする。

## 【 0 0 3 3 】

この請求項 1 2 にかかる発明によれば、バージョン管理手段がサポート可能な範囲のバージョンか否かをアプリケーションのバージョン番号に基づいて判定するようにしたため、容易かつ正確に判定することができる。

## 【 0 0 3 4 】

また、請求項 1 3 にかかる発明は、請求項 1 2 に記載の画像形成装置において、前記バージョン管理手段は、サポート可能な範囲のバージョン番号が登録されたバージョン管理テーブルをさらに備え、前記アプリケーションのバージョン番号を前記バージョン管理テーブルと比較して判定することを特徴とする。

## 【 0 0 3 5 】

この請求項 1 3 にかかる発明によれば、バージョン管理手段がサポート可能な範囲のバージョン番号が登録されたバージョン管理テーブルを備えていて、アプリケーションのバージョン番号をバージョン管理テーブルと比較して判定するようにしたため、サポート可能なアプリケーションか否かを容易かつ正確に判定することができる。

## 【 0 0 3 6 】

また、請求項 1 4 にかかる発明は、画像形成処理で使用されるハードウェア資

源と、画像形成処理にかかるユーザサービスにそれぞれ固有の処理を行うアプリケーションのプロセスと、前記アプリケーションと前記ハードウェア資源との間に介在し、前記ユーザサービスを提供する際に、アプリケーションの少なくとも2つが共通的に必要とする前記ハードウェア資源の獲得要求、管理、実行制御並びに画像形成処理を行うコントロールサービスのプロセスと、前記コントロールサービスが予め定義された関数によって前記アプリケーションからの処理要求を受信可能とするアプリケーションプログラムインタフェースとを備えた画像形成装置であって、前記アプリケーションが使用する関数を全種類抽出し、その抽出した各関数と、前記コントロールサービスのいずれかのコンポーネントとの間にバージョン差が生じたときに、そのバージョン差がサポート可能な範囲か否かを判定して、その判定結果を前記アプリケーションに通知するバージョン管理手段を備えたことを特徴とする。

## 【 0 0 3 7 】

この請求項14にかかる発明によれば、バージョン管理手段によりアプリケーションの使用関数を全種類抽出し、その抽出した各関数と、コントロールサービスとの間にバージョン差が生じた場合に、サポート可能な範囲のバージョン差か否かを判定して、その判定結果をアプリケーションに通知するようにしたため、アプリケーションが使用していない関数がバージョンアップされてもバージョン管理手段によるバージョンチェックに影響を与えないことから、サポート可能な範囲を広げることができる。また、サポートが不可能なバージョン差があると判定されても、その原因となる関数を容易に特定することができる。

## 【 0 0 3 8 】

また、請求項15にかかる発明は、画像形成処理で使用されるハードウェア資源と、画像形成処理にかかるユーザサービスにそれぞれ固有の処理を行うアプリケーションのプロセスと、前記アプリケーションと前記ハードウェア資源との間に介在し、前記ユーザサービスを提供する際に、アプリケーションの少なくとも2つが共通的に必要とする前記ハードウェア資源の獲得要求、管理、実行制御並びに画像形成処理を行うコントロールサービスのプロセスと、前記コントロールサービスが予め定義された関数によって前記アプリケーションからの処理要求を

受信可能とするアプリケーションプログラムインタフェースとを備えた画像形成装置であって、前記アプリケーションが使用する関数を全種類抽出し、その抽出した各関数と、前記アプリケーションプログラムインタフェースとの間にバージョン差が生じたときに、そのバージョン差がサポート可能な範囲か否かを判定して、その判定結果を前記アプリケーションに通知するバージョン管理手段を備えたことを特徴とする。

## 【 0 0 3 9 】

この請求項 1 5 にかかる発明によれば、バージョン管理手段によりアプリケーションの使用する関数を全種類抽出し、その抽出した各関数と、アプリケーションプログラムインタフェースとのバージョンとの間にバージョン差が生じた場合に、サポート可能な範囲のバージョン差か否かを判定して、その判定結果をアプリケーションに通知するようにしたため、アプリケーションが使用していない関数がバージョンアップされてもバージョン管理手段によるバージョンチェックに影響を与えないことから、サポート可能な範囲を広げることができる。また、サポートが不可能なバージョン差があると判定されても、その原因となる関数を容易に特定することができる。

## 【 0 0 4 0 】

また、請求項 1 6 にかかる発明は、請求項 1 4 または 1 5 に記載の画像形成装置において、前記バージョン管理手段は、サポート可能な範囲のバージョンか否かを前記アプリケーションが使用する全ての関数番号とバージョン番号とに基づいて判定することを特徴とする。

## 【 0 0 4 1 】

この請求項 1 6 にかかる発明によれば、バージョン管理手段がサポート可能な範囲のバージョンか否かを判定するにあたって、アプリケーションが使用する全ての関数の番号とバージョン番号に基づいて行われるため、単一の番号比較だけでできることから管理が容易となり、バージョンチェック処理の負荷が小さくなるという利点がある。

## 【 0 0 4 2 】

また、請求項 1 7 にかかる発明は、請求項 1 6 に記載の画像形成装置において



、前記バージョン管理手段は、サポート可能な範囲の全ての関数番号とバージョン番号とが登録されたバージョン管理テーブルをさらに備え、前記アプリケーションが使用する全ての関数番号とバージョン番号を前記バージョン管理テーブルと比較して判定することを特徴とする。

## 【 0 0 4 3 】

この請求項 1 7 にかかる発明によれば、バージョン管理手段は、バージョン管理テーブルに登録されているサポート可能な範囲の全ての関数番号とバージョン番号と、アプリケーションが使用する全ての関数番号とバージョン番号とを比較して判定するようにしたため、バージョン差の生じたアプリケーションがサポート可能か否かを容易かつ正確に判定することができる。

## 【 0 0 4 4 】

また、請求項 1 8 にかかる発明は、請求項 1 0 ～ 1 7 のいずれか一つに記載の画像形成装置において、前記コントロールサービスをサーバとしたクライアントプロセスとして動作し、かつ前記アプリケーションをクライアントとしたサーバプロセスとして動作する仮想アプリケーションサービスをさらに備え、前記バージョン管理手段は、前記仮想アプリケーションサービスに含まれていることを特徴とする。

## 【 0 0 4 5 】

この請求項 1 8 にかかる発明によれば、バージョン管理手段は、コントロールサービスをサーバとしたクライアントプロセスとして動作し、かつアプリケーションをクライアントとしたサーバプロセスとして動作する仮想アプリケーションサービスに含まれているため、バージョン差が生じた場合にバージョン差を吸収して整合性をとることができ、また、コントロールサービスからアプリケーションに対するメッセージを選択することによって、インタフェースを隠蔽して重要なインタフェースの秘匿性を確保しつつ、新規アプリケーションのプログラム開発を効率的に行うことができる。

## 【 0 0 4 6 】

また、請求項 1 9 にかかる発明は、請求項 1 8 に記載の画像形成装置において、前記バージョン管理手段は、前記仮想アプリケーションサービスのプロセス内

部でスレッドとして生成されることを特徴とする。

## 【 0 0 4 7 】

この請求項 1 9 にかかる発明によれば、バージョン管理手段は、仮想アプリケーションサービスのプロセス内部にスレッドとして生成され、実行されるため、仮想アプリケーションサービスのプロセス内部で軽快に動作させることができると共に、プロセス内部に複数個のスレッドを生成した場合であっても、それらを並行処理することができる。

## 【 0 0 4 8 】

また、請求項 2 0 にかかる発明は、画像形成処理で使用されるハードウェア資源と、画像形成処理にかかるユーザサービスにそれぞれ固有の処理を行うアプリケーションのプロセスと、前記アプリケーションと前記ハードウェア資源との間に介在し、前記ユーザサービスを提供する際に、アプリケーションの少なくとも 2 つが共通的に必要とする前記ハードウェア資源の獲得要求、管理、実行制御並びに画像形成処理を行うコントロールサービスのプロセスと、前記コントロールサービスが予め定義された関数によって前記アプリケーションからの処理要求を受信可能とするアプリケーションプログラムインタフェースとを備えた画像形成装置上で、前記アプリケーションとのバージョン差を吸収するラッピング処理方法であって、前記コントロールサービスのいずれかのコンポーネントがバージョンアップしたときに、バージョン差の生じたアプリケーションからの関数呼び出しに対して整合性をとるラッピング処理ステップを含むことを特徴とする。

## 【 0 0 4 9 】

この請求項 2 0 にかかる発明によれば、コントロールサービスのいずれかのコンポーネントがバージョンアップしたとき、ラッピング処理ステップによってバージョン差の生じたアプリケーションからの関数呼び出しに対して整合性をとるため、バージョン差が吸収され、バージョン不整合による不具合発生を防止することができる。

## 【 0 0 5 0 】

また、請求項 2 1 にかかる発明は、画像形成処理で使用されるハードウェア資源と、画像形成処理にかかるユーザサービスにそれぞれ固有の処理を行うアプリ

ケーションのプロセスと、前記アプリケーションと前記ハードウェア資源との間に介在し、前記ユーザサービスを提供する際に、アプリケーションの少なくとも2つが共通的に必要とする前記ハードウェア資源の獲得要求、管理、実行制御並びに画像形成処理を行うコントロールサービスのプロセスと、前記コントロールサービスが予め定義された関数によって前記アプリケーションからの処理要求を受信可能とするアプリケーションプログラムインタフェースとを備えた画像形成装置上で、前記アプリケーションとのバージョン差を吸収するラッピング処理方法であって、前記アプリケーションプログラムインタフェースがバージョンアップしたときに、バージョン差の生じたアプリケーションからの関数呼び出しに対して整合性をとるラッピング処理ステップを含むことを特徴とする。

## 【0051】

この請求項21にかかる発明によれば、アプリケーションプログラムインタフェースがバージョンアップしたとき、ラッピング処理ステップによってバージョン差の生じたアプリケーションからの関数呼び出しに対して整合性をとるようにしたため、バージョン差が吸収され、バージョン不整合による不具合発生を防止することができる。

## 【0052】

また、請求項22にかかる発明は、画像形成処理で使用されるハードウェア資源と、画像形成処理にかかるユーザサービスにそれぞれ固有の処理を行うアプリケーションのプロセスと、前記アプリケーションと前記ハードウェア資源との間に介在し、前記ユーザサービスを提供する際に、アプリケーションの少なくとも2つが共通的に必要とする前記ハードウェア資源の獲得要求、管理、実行制御並びに画像形成処理を行うコントロールサービスのプロセスと、前記コントロールサービスが予め定義された関数によって前記アプリケーションからの処理要求を受信可能とするアプリケーションプログラムインタフェースとを備えた画像形成装置上で、前記アプリケーションとのバージョン差を吸収するラッピング処理方法であって、前記コントロールサービスから前記アプリケーションに対するメッセージを取捨選択するラッピング処理ステップを含むことを特徴とする。

## 【0053】

この請求項 2 2 にかかる発明によれば、ラッピング処理ステップによってコントロールサービスからアプリケーションに対するメッセージを取捨選択するため、サードベンダなどの第三者に対してコントロールサービスとの間のインタフェースを隠蔽することで、重要なインタフェースの秘匿性を確保し、第三者からコントロールサービスに直接アクセスされるのを回避して、画像形成装置におけるセキュリティ面を確保することにより、障害の発生が未然に防止できる。

## 【 0 0 5 4 】

また、請求項 2 3 にかかる発明は、請求項 2 0 または 2 1 に記載のラッピング処理方法において、前記ラッピング処理ステップは、関数または関数の引数が追加されてバージョンアップしたときに、追加された関数または関数の引数に対応する機能を見捨てして整合性を見とることを特徴とする。

## 【 0 0 5 5 】

この請求項 2 3 にかかる発明によれば、ラッピング処理ステップにおいて、関数または関数の引数が追加されてバージョンアップした場合、追加された関数または関数の引数に対応する機能を見捨てして整合性を見とるようにしたため、アプリケーションからの関数呼び出しにされる関数または関数の引数をそのまま使うことができ、アプリケーションを再コンパイルすることなくバージョン差を吸収することができる。

## 【 0 0 5 6 】

また、請求項 2 4 にかかる発明は、請求項 2 0 または 2 1 に記載のラッピング処理方法において、前記ラッピング処理ステップは、関数または関数の引数が分割されてバージョンアップしたときに、分割された関数または関数の引数に応じてダミーの関数または関数の引数を補填して整合性を見とることを特徴とする。

## 【 0 0 5 7 】

この請求項 2 4 にかかる発明によれば、ラッピング処理ステップにおいて、関数または関数の引数が分割されてバージョンアップした場合、分割された関数または関数の引数に応じてダミーの関数または関数の引数を補填して整合性を見とるようにしたため、関数の対応関係を維持することが可能となり、アプリケーションを再コンパイルすることなくバージョン差を吸収することができる。

## 【 0 0 5 8 】

また、請求項 2 5 にかかる発明は、請求項 2 2 に記載のラッピング処理方法において、前記ラッピング処理ステップは、前記アプリケーションに対して通知しない特定のメッセージを予め設定可能にしたことを特徴とする。

## 【 0 0 5 9 】

この請求項 2 5 にかかる発明によれば、ラッピング処理ステップにおいて、アプリケーションに対して通知しない特定のメッセージが予め設定可能なため、サードベンダなどに公開するインタフェースの基準を任意に決定することができ、重要なインタフェースの秘匿性を確保することができる。

## 【 0 0 6 0 】

また、請求項 2 6 にかかる発明は、画像形成処理で使用されるハードウェア資源と、画像形成処理にかかるユーザサービスにそれぞれ固有の処理を行うアプリケーションのプロセスと、前記アプリケーションと前記ハードウェア資源との間に介在し、前記ユーザサービスを提供する際に、アプリケーションの少なくとも 2 つが共通的に必要とする前記ハードウェア資源の獲得要求、管理、実行制御並びに画像形成処理を行うコントロールサービスのプロセスと、前記コントロールサービスが予め定義された関数により前記アプリケーションからの処理要求を受信可能とするアプリケーションプログラムインタフェースとを備えた画像形成装置上で、前記アプリケーションとの間のバージョンを管理するバージョン管理方法であって、前記アプリケーションと前記コントロールサービスのいずれかのコンポーネントとの間にバージョン差が生じたときに、そのバージョン差がサポート可能な範囲か否かを判定して、その判定結果を前記アプリケーションに通知するバージョン管理ステップを含むことを特徴とする。

## 【 0 0 6 1 】

この請求項 2 6 にかかる発明によれば、アプリケーションとコントロールサービスのいずれかのコンポーネントとの間でバージョン差が生じた場合に、バージョン管理ステップによりバージョン差がサポート可能な範囲か否かを判定し、その判定結果をアプリケーションに通知するようにしたため、例えばアプリケーションの登録時などにバージョンの整合性を判定すれば、アプリケーションの実行

前にバージョンの適合性の有無が判断でき、障害の発生を未然に防止することができる。

## 【 0 0 6 2 】

また、請求項 2 7 にかかる発明は、画像形成処理で使用するハードウェア資源と、画像形成処理にかかるユーザサービスにそれぞれ固有の処理を行うアプリケーションのプロセスと、前記アプリケーションと前記ハードウェア資源との間に介在し、前記ユーザサービスを提供する際に、アプリケーションの少なくとも 2 つが共通的に必要とする前記ハードウェア資源の獲得要求、管理、実行制御並びに画像形成処理を行うコントロールサービスのプロセスと、前記コントロールサービスが予め定義された関数により前記アプリケーションからの処理要求を受信可能とするアプリケーションプログラムインタフェースとを備えた画像形成装置上で、前記アプリケーションとの間のバージョンを管理するバージョン管理方法であって、前記アプリケーションと前記アプリケーションプログラムインタフェースとの間にバージョン差が生じたときに、そのバージョン差がサポート可能な範囲か否かを判定して、その判定結果を前記アプリケーションに通知するバージョン管理ステップを含むことを特徴とする。

## 【 0 0 6 3 】

この請求項 2 7 にかかる発明によれば、アプリケーションとアプリケーションプログラムインタフェースとの間でバージョン差が生じた場合に、バージョン管理ステップによりバージョン差がサポート可能な範囲か否かを判定し、その判定結果をアプリケーションに通知するようにしたため、例えばアプリケーションの登録時などにバージョンの整合性を判定すれば、アプリケーションの実行前にバージョンの適合性の有無が判断でき、障害の発生を未然に防止することができる。

## 【 0 0 6 4 】

また、請求項 2 8 にかかる発明は、請求項 2 6 または 2 7 に記載のバージョン管理方法において、前記バージョン管理ステップは、サポート可能な範囲のバージョン差か否かを前記アプリケーションのバージョン番号に基づいて判定することを特徴とする。

## 【 0 0 6 5 】

この請求項 2 8 にかかる発明によれば、バージョン管理ステップにおいて、サポート可能な範囲のバージョンか否かをアプリケーションのバージョン番号に基づいて判定するようにしたため、容易かつ正確に判定することができる。

## 【 0 0 6 6 】

また、請求項 2 9 にかかる発明は、請求項 2 8 に記載のバージョン管理方法において、前記バージョン管理ステップは、サポート可能な範囲のバージョン番号が登録されたバージョン管理テーブルと比較して判定することを特徴とする。

## 【 0 0 6 7 】

この請求項 2 9 にかかる発明によれば、バージョン管理ステップにおいて、サポート可能な範囲のバージョン番号が登録されたバージョン管理テーブルを用い、登録されるアプリケーションのバージョン番号をバージョン管理テーブルと比較して判定するようにしたため、サポート可能なアプリケーションか否かを容易かつ正確に判定することができる。

## 【 0 0 6 8 】

また、請求項 3 0 にかかる発明は、画像形成処理で使用されるハードウェア資源と、画像形成処理にかかるユーザサービスにそれぞれ固有の処理を行うアプリケーションのプロセスと、前記アプリケーションと前記ハードウェア資源との間に介在し、前記ユーザサービスを提供する際に、アプリケーションの少なくとも 2 つが共通的に必要とする前記ハードウェア資源の獲得要求、管理、実行制御並びに画像形成処理を行うコントロールサービスのプロセスと、前記コントロールサービスが予め定義された関数により前記アプリケーションからの処理要求を受信可能とするアプリケーションプログラムインタフェースとを備えた画像形成装置上で、前記アプリケーションが使用する全関数についてサポートが可能か否かを判定するバージョン管理方法であって、前記アプリケーションが使用する関数を全種類抽出し、その抽出した各関数と、前記コントロールサービスのいずれかのコンポーネントとの間にバージョン差が生じたときに、そのバージョン差がサポート可能な範囲か否かを判定して、その判定結果を前記アプリケーションに通知するバージョン管理ステップを含むことを特徴とする。

## 【 0 0 6 9 】

この請求項 3 0 にかかる発明によれば、バージョン管理ステップによりアプリケーションの使用する関数を全種類抽出し、その抽出した各関数と、コントロールサービスとの間にバージョン差が生じた場合に、サポート可能な範囲のバージョン差か否かを判定して、その判定結果をアプリケーションに通知するようにしたため、アプリケーションが使用していない関数がバージョンアップされてもバージョン管理ステップによるバージョンチェックに影響を与えないことから、サポート可能な範囲を広げることができる。また、サポートが不可能なバージョン差があると判定されても、その原因となる関数を容易に特定することができる。

## 【 0 0 7 0 】

また、請求項 3 1 にかかる発明は、画像形成処理で使用されるハードウェア資源と、画像形成処理にかかるユーザサービスにそれぞれ固有の処理を行うアプリケーションのプロセスと、前記アプリケーションと前記ハードウェア資源との間に介在し、前記ユーザサービスを提供する際に、アプリケーションの少なくとも 2 つが共通的に必要とする前記ハードウェア資源の獲得要求、管理、実行制御並びに画像形成処理を行うコントロールサービスのプロセスと、前記コントロールサービスが予め定義された関数により前記アプリケーションからの処理要求を受信可能とするアプリケーションプログラムインタフェースとを備えた画像形成装置上で、前記アプリケーションが使用する全関数についてサポートが可能か否かを判定するバージョン管理方法であって、前記アプリケーションが使用する関数を全種類抽出し、その抽出した各関数と、前記アプリケーションプログラムインタフェースとの間にバージョン差が生じたときに、そのバージョン差がサポート可能な範囲か否かを判定して、その判定結果を前記アプリケーションに通知するバージョン管理ステップを含むことを特徴とする。

## 【 0 0 7 1 】

この請求項 3 1 にかかる発明によれば、バージョン管理ステップによりアプリケーションの使用する関数を全種類抽出し、その抽出した各関数と、アプリケーションプログラムインタフェースとのバージョンとの間にバージョン差が生じた場合に、サポート可能な範囲のバージョン差か否かを判定して、その判定結果を



アプリケーションに通知するようにしたため、アプリケーションが使用していない関数がバージョンアップされてもバージョン管理ステップによるバージョンチェックに影響を与えないことから、サポート可能な範囲を広げることができる。また、サポートが不可能なバージョン差があると判定されても、その原因となる関数を容易に特定することができる。

## 【 0 0 7 2 】

また、請求項 3 2 にかかる発明は、請求項 3 0 または 3 1 に記載のバージョン管理方法において、前記バージョン管理ステップは、サポート可能な範囲のバージョン差か否かを前記アプリケーションが使用する全ての関数番号とバージョン番号とに基づいて判定することを特徴とする。

## 【 0 0 7 3 】

この請求項 3 2 にかかる発明によれば、バージョン管理ステップによりサポート可能な範囲のバージョンか否かを判定するにあたって、アプリケーションが使用する全ての関数の番号とバージョン番号に基づいて行われるため、単一の番号比較だけでできることから管理が容易となり、バージョンチェック処理の負荷が小さくなるという利点がある。

## 【 0 0 7 4 】

また、請求項 3 3 にかかる発明は、請求項 3 2 に記載のバージョン管理方法において、前記バージョン管理ステップは、前記アプリケーションが使用する全ての関数番号とバージョン番号を、サポート可能な範囲の関数番号とバージョン番号が登録されたバージョン管理テーブルと比較して判定することを特徴とする。

## 【 0 0 7 5 】

この請求項 3 3 にかかる発明によれば、バージョン管理ステップでは、バージョン管理テーブルに登録されているサポート可能な範囲の全ての関数番号とバージョン番号と、アプリケーションが使用する全ての関数番号とバージョン番号とを比較して判定するようにしたため、バージョン差の生じたアプリケーションがサポート可能か否かを容易かつ正確に判定することができる。

## 【 0 0 7 6 】

## 【発明の実施の形態】

以下に添付図面を参照して、この発明にかかる画像形成装置およびラッピング処理方法並びにバージョン管理方法の好適な実施の形態を詳細に説明する。

【0077】

(実施の形態1)

図1は、この発明の実施の形態1である画像形成装置（以下、「複合機」という）の構成を示すブロック図である。図1に示すように、複合機100は、白黒レーザプリンタ（B&W LP）101と、カラーレーザプリンタ（Color LP）102と、スキャナ、ファクシミリ、ハードディスク、メモリ、ネットワークインタフェースなどのハードウェアリソース103を有するとともに、プラットフォーム120とアプリケーション130と仮想アプリケーションサービス（VAS：Virtual Application Service）140から構成されるソフトウェア群110とを備えている。

【0078】

仮想アプリケーションサービス（VAS）140は、アプリケーション130とプラットフォーム120の間に配置される。VAS140は、アプリケーション130の各アプリが初めて登録されるときに、登録処理が行われ、アプリから見るとプラットフォーム120のサービス層として認識され、サービス層から見るとアプリとして認識されるように登録される。このVAS140の第1の基本機能としては、プラットフォーム120の各種コントロールサービスあるいはそのアプリケーションプログラムインタフェース（API）が仕様変更などでバージョンアップされた場合に、バージョン差の生じたアプリケーション130を再コンパイルすることなくそのまま関数呼び出しが行えるよう、バージョン差を吸収して整合性を保つと共に、コントロールサービスからのメッセージを取捨選択することによってプラットフォーム120を意図的に隠蔽することができるラッピング機能を備えている。

【0079】

また、このVAS140の第2の基本機能としては、各アプリが登録される際に、各アプリとコントロールサービスおよびAPIのそれぞれのバージョンを検出し、その検出されたバージョン差がVAS140によってサポート可能な範囲

か否かを判定して、その判定結果をアプリに通知することにより、アプリ動作前にバージョンが不整合かどうかを判明するバージョン管理機能を備えている。

## 【 0 0 8 0 】

プラットフォーム 1 2 0 は、アプリケーションからの処理要求を解釈してハードウェア資源の獲得要求を発生させるコントロールサービスと、一または複数のハードウェア資源の管理を行い、コントロールサービスからの獲得要求を調停するシステムリソースマネージャ (SRM) 1 2 3 と、汎用 OS 1 2 1 とを有する。

## 【 0 0 8 1 】

コントロールサービスは、複数のサービスモジュールから形成され、SCS (システムコントロールサービス) 1 2 2 と、ECS (エンジンコントロールサービス) 1 2 4 と、MCS (メモリコントロールサービス) 1 2 5 と、OCS (オペレーションパネルコントロールサービス) 1 2 6 と、FCS (ファックスコントロールサービス) 1 2 7 と、NCS (ネットワークコントロールサービス) 1 2 8 とから構成される。なお、このプラットフォーム 1 2 0 は、予め定義された関数により前記アプリケーション 1 3 0 から処理要求を受信可能とするアプリケーションプログラムインタフェース (API) を有している。

## 【 0 0 8 2 】

汎用 OS 1 2 1 は、UNIX (登録商標) などの汎用オペレーティングシステムであり、プラットフォーム 1 2 0 並びにアプリケーション 1 3 0 の各ソフトウェアをそれぞれプロセスとして並列実行する。

## 【 0 0 8 3 】

SRM 1 2 3 のプロセスは、SCS 1 2 2 とともにシステムの制御およびリソースの管理を行うものである。SRM 1 2 3 のプロセスは、スキャナ部やプリンタ部などのエンジン、メモリ、HDD ファイル、ホスト I / O (セントロ I / F、ネットワーク I / F、IEEE 1 3 9 4 I / F、RS 2 3 2 C I / F など) のハードウェア資源を利用する上位層からの要求にしたがって調停を行い、実行制御する。

## 【 0 0 8 4 】

具体的には、この SRM 1 2 3 は、要求されたハードウェア資源が利用可能で

あるか（他の要求により利用されていないかどうか）を判断し、利用可能であれば要求されたハードウェア資源が利用可能である旨を上位層に伝える。また、S R M 1 2 3 は、上位層からの要求に対してハードウェア資源の利用スケジューリングを行い、要求内容（例えば、プリンタエンジンにより紙搬送と作像動作、メモリ確保、ファイル生成など）を直接実施している。

## 【 0 0 8 5 】

S C S 1 2 2 のプロセスは、アプリ管理、操作部制御、システム画面表示、L E D 表示、リソース管理、割り込みアプリ制御などを行う。

## 【 0 0 8 6 】

E C S 1 2 4 のプロセスは、白黒レーザプリンタ（B&W LP）1 0 1、カラーレーザプリンタ（Color LP）1 0 2、スキャナ、ファクシミリなどからなるハードウェアリソース1 0 3のエンジンの制御を行う。

## 【 0 0 8 7 】

M C S 1 2 5 のプロセスは、画像メモリの取得および解放、ハードディスク装置（HDD）の利用、画像データの圧縮および伸張などを行う。

## 【 0 0 8 8 】

F C S 1 2 7 のプロセスは、システムコントローラの各アプリ層からP S T N / I S D N 網を利用したファクシミリ送受信、B K M（バックアップS R A M）で管理されている各種ファクシミリデータの登録／引用、ファクシミリ読みとり、ファクシミリ受信印刷、融合送受信を行うためのA P Iを提供する。

## 【 0 0 8 9 】

N C S 1 2 8 のプロセスは、ネットワークI / Oを必要とするアプリケーションに対して共通に利用できるサービスを提供するためのプロセスであり、ネットワーク側から各プロトコルによって受信したデータを各アプリケーションに振り分けたり、アプリケーションからデータをネットワーク側に送信する際の仲介を行う。具体的には、ftpd、httpd、lpd、snmpd、telnetd、smtpdなどのサーバデーモンや、同プロトコルのクライアント機能などを有している。

## 【 0 0 9 0 】

O C S 1 2 6 のプロセスは、オペレータ（ユーザ）と本体制御間の情報伝達手

段となるオペレーションパネル（操作パネル）の制御を行う。OCS126は、オペレーションパネルからキー押下をキーイベントとして取得し、取得したキーに対応したキーイベント関数をSCS122に送信するOCSプロセスの部分と、アプリケーション130またはコントロールサービスからの要求によりオペレーションパネルに各種画面を描画出力する描画関数やその他オペレーションパネルに対する制御を行う関数などが予め登録されたOCSライブラリの部分とから構成される。このOCSライブラリは、アプリケーション130およびコントロールサービスの各モジュールにリンクされて実装されている。なお、OCS126のすべてをプロセスとして動作させるように構成しても良く、あるいはOCS126のすべてをOCSライブラリとして構成しても良い。

#### 【0091】

アプリケーション130は、ページ記述言語（PDL）、PCLおよびポストスクリプト（PS）を有するプリンタ用のアプリケーションであるプリンタアプリ111と、コピー用アプリケーションであるコピーアプリ112と、ファクシミリ用アプリケーションであるファックスアプリ113と、スキャナ用アプリケーションであるスキャナアプリ114と、ネットワークファイル用アプリケーションであるネットファイルアプリ115と、工程検査用アプリケーションである工程検査アプリ116とを有している。これらの各アプリは、その起動時にVAS140に対して自プロセスのプロセスIDとともにアプリ登録要求メッセージを送信し、アプリ登録要求メッセージを受信したVAS140によって、起動したアプリに対する登録が行われるようになっている。

#### 【0092】

アプリケーション130の各プロセス、コントロールサービスの各プロセスは、関数呼び出しとその戻り値送信およびメッセージの送受信によってプロセス間通信を行いながら、コピー、プリンタ、スキャナ、ファクシミリなどの画像形成処理にかかるユーザサービスを実現している。

#### 【0093】

このように、実施の形態1にかかる複合機100には、複数のアプリケーション130および複数のコントロールサービスが存在し、いずれもプロセスとして

動作している。そして、これらの各プロセス内部には、一または複数のスレッドが生成されて、スレッド単位の並列実行が行われる。そして、コントロールサービスがアプリケーション 1 3 0 に対し共通サービスを提供しており、このため、これらの多数のプロセスが並列動作、およびスレッドの並列動作を行って互いにプロセス間通信を行って協調動作をしながら、コピー、プリンタ、スキャナ、ファクシミリなどの画像形成処理にかかるユーザサービスを提供するようになっている。また、複合機 1 0 0 には、サードベンダなどの第三者がコントロールサービス層の上のアプリケーション層に新規アプリ 1 1 7、1 1 8 を開発して搭載することが可能となっている。図 1 では、この新規アプリ 1 1 7、1 1 8 を搭載した例を示している。

## 【 0 0 9 4 】

なお、実施の形態 1 にかかる複合機 1 0 0 では、複数のアプリケーション 1 3 0 のプロセスと複数のコントロールサービスのプロセスとが動作しているが、アプリケーション 1 3 0 とコントロールサービスのプロセスをそれぞれ単一の構成とすることも可能である。また、各アプリケーション 1 3 0 は、アプリケーションごとに追加または削除することができる。さらに、実施の形態 1 にかかる複合機 1 0 0 では、V A S 1 4 0 に加えて動的リンク手段としてのダイナミックリンクライブラリ ( D L L ) を採用することが可能である。

## 【 0 0 9 5 】

図 2 は、実施の形態 1 にかかる複合機 1 0 0 の V A S 1 4 0 の構成と、V A S 1 4 0 と各アプリ、コントロールサービス層 1 5 0 および汎用 O S 1 2 1 との関係を示すブロック図である。なお、図 2 では、アプリケーション 1 3 0 の例として、プリンタアプリ 1 1 1、コピーアプリ 1 1 2、新規アプリ 1 1 7、1 1 8 を示しているが、他のアプリでも同様の構成である。

## 【 0 0 9 6 】

仮想アプリケーションサービス ( V A S ) 1 4 0 のプロセスには、ディスパッチャ 1 4 4 と、制御スレッド 1 4 3 と、ラッピング処理スレッド 1 4 1 と、バージョン管理スレッド 1 4 2 とが動作している。

## 【 0 0 9 7 】

ディスパッチャ 1 4 4 は、アプリケーション 1 3 0 やコントロールサービスからのメッセージ受信を監視し、受信したメッセージに応じて制御スレッド 1 4 3、ラッピング処理スレッド 1 4 1、バージョン管理スレッド 1 4 2 に処理要求を行うものである。実施の形態 1 の複合機 1 0 0 では、ディスパッチャ 1 4 4 は、各アプリが起動したときにアプリからのアプリ登録要求メッセージを受信すると、そのアプリ登録要求メッセージを制御スレッド 1 4 3 に送信するようになっている。

## 【 0 0 9 8 】

制御スレッド 1 4 3 は、ディスパッチャ 1 4 4 からアプリ登録要求メッセージを受信してアプリ登録処理を行う。ここで、アプリ登録処理とは、RAM 2 1 0 にアプリ登録テーブル（図示せず）を生成し、アプリ登録要求メッセージを送信したアプリの識別情報であるアプリ ID をアプリ登録テーブルに記録する処理をいう。

## 【 0 0 9 9 】

また、制御スレッド 1 4 3 は、HD 2 0 0 に格納されたラッピング処理情報ファイル 2 0 1 を参照して、アプリ登録要求を行ったアプリについて、ラッピング処理情報が記録されているか否かをチェックする。

## 【 0 1 0 0 】

そして、ラッピング処理手段としてのラッピング処理スレッド 1 4 1 は、コントロールサービスあるいはその上の API の関数または関数の引数が追加されてバージョンアップした場合、制御スレッド 1 4 3 からの処理要求を受け、アプリからの関数呼び出しに対して対応する関数内で実行されるため、追加された機能が無視する（使えないとする）ならばバージョンの不整合は起こらない。これは、VAS 1 4 0 上に設けられた不図示のインタフェース（VAS-API）は、コントロールサービスあるいはその上の API をバージョンアップしたとしてもアプリ間での整合性が保持されるため、アプリを再コンパイルする必要はなくなる。

## 【 0 1 0 1 】

また、コントロールサービスあるいはその上の API の関数または関数の引数

が分割されてバージョンアップした場合は、ラッピング処理情報ファイル 2 0 1 を参照して対応するダミーの関数または関数の引数を補填することにより、アプリからの関数呼び出しに対して対応関数を起動させることが可能となり、V A S 1 4 0 がバージョン差を吸収することから、アプリを再コンパイルする必要がなくなる。

#### 【 0 1 0 2 】

さらに、ラッピング処理スレッド 1 4 1 は、コントロールサービス層 1 5 0 からアプリに対して行われるメッセージ通知を選択的行えるよう、予めラッピング処理情報ファイル 2 0 1 に設定することが可能である。特に、複合機 1 0 0 のシステムに大きく影響を与えるインタフェースについては、サードベンダなどの第三者に対し隠蔽しておき、直接コントロールサービスに対してアクセスすることを回避することが、複合機のセキュリティ面および障害発生 of 未然防止の面から好ましい。

#### 【 0 1 0 3 】

また、バージョン管理手段としてのバージョン管理スレッド 1 4 2 は、制御スレッド 1 4 3 からの処理要求を受け、登録されたアプリのバージョンとコントロールサービスのいずれかのコンポーネントおよび A P I とのバージョンの違いを検出すると、サポート可能な範囲のバージョン番号などが登録されている R A M 2 1 0 のバージョン管理テーブル 2 1 1 と比較することにより、発生したバージョン差が V A S 1 4 0 によってサポート可能な範囲か否かを判定することができる。かかるサポート可能な範囲のバージョン番号などが登録されたバージョン管理情報は、アプリごとのレコードとして記録される。

#### 【 0 1 0 4 】

図 3 は、H D 2 0 0 に格納されるラッピング処理情報ファイル 2 0 1 の内容例を示す説明図である。図 3 に示すように、ラッピング処理情報ファイル 2 0 1 は、コントロールサービス層 1 5 0 からアプリに対するメッセージ（イベント）の非通知設定を行うための情報ファイルであり、複合機 1 0 0 のシステムに大きく影響を与えるインタフェースについては第三者に対して隠蔽するよう、メッセージごとに任意の非通知設定を行うことができる。図 3 のイベント A と C は、非通



知設定がなされているのでアプリには通知されず、また、イベントBのように非通知設定がなされていないイベントについては、通常通り通知が行われる。

## 【0105】

図4は、RAM210に格納される一括管理用のバージョン管理テーブル211の内容例を示す説明図である。図4に示すように、バージョン管理テーブル211には、VAS140でサポート範囲内とする各アプリ（アプリID）のバージョン番号が格納されていて、アプリ登録の際に通知されるアプリのバージョン番号をこのバージョン管理テーブル211に格納されるバージョン番号と比較することにより、サポート範囲内か否かを容易に判定することができる。ここでいうサポート範囲とは、VAS140のリリース時点より前のバージョンのことであり、VAS140によって吸収可能な一定のバージョン範囲のことをいう。

## 【0106】

図5は、RAM210に格納される関数単位管理用のバージョン管理テーブル211の内容例を示す説明図である。図5に示すように、このバージョン管理テーブル211には、関数番号とバージョン番号とそのサポート範囲とが格納されていて、アプリがVAS140に対して使用する全種類の関数を抽出し、各関数をバージョン管理テーブル211と比較することにより、関数単位でサポートが可能か否かを判定することができる。

## 【0107】

次に、このように構成された複合機100のVAS140によるラッピング処理とバージョン管理について説明する。図6は、VAS140の制御スレッド143によりバージョン差を吸収するためのラッピング処理を行う手順を示すフローチャートである。

## 【0108】

図6において、アプリとコントロールサービスおよびそのAPIのバージョンが一致している間は、障害は発生しないが、コントロールサービスあるいはそのAPIがバージョンアップされてアプリとの間にバージョン差が生じた場合、バージョン不整合が問題となる。ここで、アプリのバージョンとは、アプリのコンパイル時に使用したVASヘッダファイルに定義されたバージョンのことをいう

。そこで、ステップS 6 0 1において、コントロールサービスおよびそのAPIがバージョンアップされた場合、そのバージョンアップの態様を判別する（ステップS 6 0 2）。例えば、関数または関数の引数の追加、削除、変更、分割、統合などによるバージョンアップが考えられるが、関数または関数の引数が削除、変更、統合された場合は、対応する関数が存在しなくなるため、アプリからの関数呼び出しがあっても整合性をとることができない。このため、アプリに対してNGであることを通知し（ステップS 6 0 6）、終了処理する（ステップS 6 0 7）。

## 【0109】

一方、ステップS 6 0 2において、関数または関数の引数の追加あるいは分割によるバージョンアップの場合、さらに追加か分割かを判別して（ステップS 6 0 3）、追加の場合は既存の関数自体は変化しないことから、追加された機能は使えないが、関数呼び出しによるバージョンの整合性を保つことができるので（ステップS 6 0 8）、そのまま通常に処理することができる（ステップS 6 0 5）。

## 【0110】

再び、ステップS 6 0 3に戻って、分割の場合は、対応する関数または関数の引数が分割されて増えるため、増えた部分にダミー関数、または、ダミー引数を補填して対応関係をとることにより（ステップS 6 0 4）、通常の処理（ステップS 6 0 5）が行えるようになる。

## 【0111】

図7は、VAS 1 4 0の制御スレッド1 4 3によりコントロールサービスの特定のインタフェースを隠蔽するためのラッピング処理を行う手順を示すフローチャートである。

## 【0112】

図7において、コントロールサービスからアプリに対して全てメッセージ通知（例えば、イベント通知）が行われるとすると、複合機1 0 0のシステムに大きく影響を与えるコントロールサービスに対して直接アクセスすることが可能となり、複合機のセキュリティ面および障害発生を未然に防止する面からも好ましく

ない。そこで、実施の形態1では、非通知にできるイベントを予め設定することができる。図7に示すように、非通知のイベントを設定する場合は（ステップS701）、特定のイベントを指定してラッピング処理情報ファイル201に対して非通知設定を行う（ステップS702）。

#### 【0113】

そして、図2のコントロールサービス層150からアプリに向けてメッセージ通知がなされる場合、ラッピング処理スレッド141によりラッピング処理情報ファイル201を参照し、非通知設定がなされていればアプリに通知せずに隠蔽する（ステップS704）。また、上記ステップS701において非通知設定自体が行われていないか、ステップS703において非通知設定されていないイベントがある場合は、通常通りアプリにメッセージが通知される（ステップS705）。ここでは、特定のイベントをアプリに通知するか否かを任意かつ事前に設定できるようにしたため、複合機のセキュリティ面および障害発生を未然に防止することが可能になる。また、実施の形態1では、新規アプリケーションを開発するサードベンダなどの第三者に特定のインタフェースを開示する必要があるため、開示できるインタフェースを自由に選択できるようにすることは非常に重要である。

#### 【0114】

図8は、VAS140の制御スレッド143によりアプリ実行の初期段階でバージョン不整合があるか否かを判定するバージョン管理のための手順を示すフローチャートである。

#### 【0115】

まず、VAS140によるバージョンチェック時期としては、アプリが実行される前であれば良く、例えばここではアプリ登録時に行ったが、これに限定されない。そこで、図2のディスパッチャ144が起動されたアプリからアプリ登録要求メッセージを受信すると、アプリ登録要求メッセージをそのアプリのプロセスIDとともに制御スレッド143に受け渡す。制御スレッド143は、アプリ登録要求メッセージをプロセスIDをディスパッチャ144から受信すると、アプリを識別するアプリIDを決定して、RAM210などに格納したアプリ登録

テーブル（図示せず）にアプリIDを記録することにより、アプリ登録が行われる。なお、アプリIDは、コピーアプリ112、プリンタアプリ111など既存のアプリケーションについては、予め定められており、各アプリIDをVAS140の内部で保持している。また、サードベンダなどが開発した新規アプリ117、118については、最初の起動時におけるアプリ登録処理の中で決定される。

## 【0116】

このようにして、アプリ登録があった場合（ステップS801）、アプリごとに持っている一種類のバージョン番号をVAS140に通知する（ステップS802）。VAS140では、RAM210のバージョン管理テーブル211（図4参照）とアプリのバージョン番号とを比較し、サポート範囲内か否かを判断する（ステップS803）。例えば、新規アプリのアプリIDが図4中の101とすると、そのサポート範囲は、バージョン1.0～1.6であるので、この範囲内のバージョンを持った新規アプリであればサポート範囲内と判定することができる。その判定結果は、当該アプリに対して「OK」が通知されると（ステップS804）、正常にアプリ登録され（ステップS805）、通常の複合機100における処理が行われる（ステップS806）。また、ステップS803でアプリのバージョン番号がバージョン管理テーブル211のサポート範囲外と判定されると、当該アプリに対して「NG」が通知され（ステップS807）、アプリ登録が終了する（ステップS808）。

## 【0117】

この図8の例では、アプリのバージョン番号を用いてバージョンを一括管理するため、アプリのバージョン番号をバージョン管理テーブル211で単純に番号比較することから、容易に管理することができ、バージョンチェック処理に要する負荷が小さくて済む利点がある。

## 【0118】

図9は、VAS140の制御スレッド143によりアプリが実際に使用している関数を全種類抽出し関数単位でバージョンに不整合があるか否かを判定するバージョン管理の手順を示すフローチャートである。

## 【 0 1 1 9 】

図 9 の場合も、図 8 と同様に V A S 1 4 0 によるバージョンチェック時期として、アプリが実行される前であれば良く、ここではアプリ登録時を想定している。このようなアプリ登録があった場合（ステップ S 9 0 1）、アプリが V A S 1 4 0 に対して使用している全種類の関数を抽出し（ステップ S 9 0 2）、その抽出された関数番号とそのバージョン番号を V A S 1 4 0 に対して使用可能か否か問い合わせる。V A S 1 4 0 では、R A M 2 1 0 のバージョン管理テーブル 2 1 1（図 5 参照）とアプリが使用している全ての関数番号とバージョン番号とを比較することにより、サポート範囲内か否かを判断することができる（ステップ S 9 0 3）。例えば、ここでは新規アプリが使用している関数番号が「2」、「3」であり、そのバージョン番号が「バージョン 1. 2」と「バージョン 2. 3」であって、それぞれの関数番号に対応したサポート範囲もバージョン管理テーブル 2 1 1 に格納されているため、関数ごとの判定結果を全てチェックした後、その判定結果を当該アプリに対して通知する。図 9 の場合、ステップ S 9 0 3 においてサポート範囲内と判定された判定結果は、上記した図 8 の A 以降と同様に処理され、ステップ S 9 0 3 においてサポート範囲内でないと判定された場合は、図 8 の B 以降と同様に処理されるため、ここでは重複説明を省略する。

## 【 0 1 2 0 】

この図 9 の例では、アプリが V A S 1 4 0 に対して実際に使用している関数番号とそのバージョン番号とを全て抽出し、バージョン管理テーブル 2 1 1 に対して関数ごとに比較するため、仮にアプリが実際に使用していない関数のバージョンが変更されたとしても、バージョンチェック結果に影響が出ず、実質的な判定結果が得られることから、より適応範囲を広げることができる。また、個々の関数でサポート範囲外と判定されて使用不可となった場合でも、その原因となった関数が容易に特定できるため、迅速に対応することができる。

## 【 0 1 2 1 】

このように、実施の形態 1 にかかる複合機 1 0 0 では、仮想アプリケーションサービス 1 4 0 のラッピング処理スレッド 1 4 1 により、バージョン差を吸収して整合性を保持することができるため、コントロールサービスとその A P I がバ

ージョンアップしたとしてもその上のアプリをコンパイルし直す必要が無くなる。また、ラッピング処理スレッド141は、コントロールサービスからアプリに対するメッセージを取捨選択できるように事前に設定することができるため、複合機100のシステムに大きく影響を与えるようなインタフェースについては、第三者に対して隠蔽することで、直接コントロールサービスに対してアクセスすることが回避でき、複合機のセキュリティ面および障害発生の未然防止の面から好ましい。また、実施の形態1にかかる複合機100では、仮想アプリケーションサービス140のバージョン管理スレッド142により、アプリのバージョン番号を用いてバージョン番号をバージョン管理テーブル211を用いて番号比較することにより、容易に管理することができる上、バージョンチェック処理の負荷が小さくて済む。さらに、仮想アプリケーションサービス140のバージョン管理スレッド142により、アプリがVAS140に対して実際に使用している関数番号とそのバージョン番号とを全て抽出して、バージョン管理テーブル211に対して関数ごとに比較を行うため、関数が変更されたことによる影響が少なくなり、実質的に適応範囲が広がり、個々の関数比較によりサポート範囲外と判定された場合でも、原因の関数を容易に特定することができるため、迅速な対応が可能となる。また、実施の形態1にかかる複合機100では、上記したようにVAS140に加えてダイナミックリンクライブラリ(DLL)を採用することも可能であり、その場合はアプリケーションとコントロールサービスあるいはインタフェース間のバージョン差の吸収力が増大し、より柔軟に整合性がとれるようになると共に、インタフェースを隠蔽することができるためインタフェースの秘匿性をより強固に確保することが可能となる。

## 【0122】

## (実施の形態2)

実施の形態1にかかる複合機100は、VAS140が全アプリケーションに対して1つのみ存在するものであった、この実施の形態2にかかる複合機では、各アプリごとに一つのVASが起動し、各VASは対応するアプリに対してのみバージョン管理およびラッピング処理を行うものである。

## 【0123】

図 1 0 は、実施の形態 2 にかかる複合機 8 0 0 の構成を示すブロック図である。図 1 0 に示すように、複合機 8 0 0 では、複数の仮想アプリケーションサービス ( V A S ) 8 4 1 ~ 8 4 8 がアプリケーション 1 3 0 の各アプリごとに動作している点が、実施の形態 1 にかかる複合機 1 0 0 と異なっている。

#### 【 0 1 2 4 】

V A S 8 4 1 ~ 8 4 8 は、プリンタアプリ 1 1 1、コピーアプリ 1 1 2、ファックスアプリ 1 1 3、スキャナアプリ 1 1 4、ネットファイルアプリ 1 1 5、工程検査アプリ 1 1 6、新規アプリ 1 1 7 および 1 1 8 に対応して、バージョン管理およびラッピング処理を行うようになっている。

#### 【 0 1 2 5 】

図 1 1 は、実施の形態 2 にかかる複合機 8 0 0 の V A S 8 4 1 ~ 8 4 8 の構成と、 V A S 8 4 1 ~ 8 4 8 と各アプリ、コントロールサービス層 1 5 0 および汎用 O S 1 2 1 との関係を示すブロック図である。なお、図 1 0 では、アプリケーション 1 3 0 として、プリンタアプリ 1 1 1、コピーアプリ 1 1 2、新規アプリ 1 1 7、1 1 8 の例を示し、さらにこれら各アプリに対応した V A S 8 4 1、8 4 2、8 4 7 および 8 4 8 を例として示したが、他のアプリの場合も同様の構成となる。

#### 【 0 1 2 6 】

また、実施の形態 2 にかかる複合機 8 0 0 では、実施の形態 1 の複合機 1 0 0 と異なり、図 1 1 に示すように、各 V A S 8 4 1 ~ 8 4 8 と各アプリとの間には V A S 制御プロセス ( デーモン ) 8 0 1 が動作している。

#### 【 0 1 2 7 】

この V A S 制御プロセス ( デーモン ) 8 0 1 は、各アプリからアプリ登録要求メッセージを受信してアプリ登録処理を行うとともに、アプリ登録要求を行ったアプリに対応した V A S 8 4 1 ~ 8 4 8 を生成する。また、 V A S 制御プロセス 8 0 1 は、 H D 2 0 0 に格納されたラッピング処理情報ファイル 2 0 1 を参照して、アプリ登録要求を行ったアプリについて、バージョン差を吸収して整合性を保つようにしたり、コントロールサービス層 1 5 0 からのメッセージを設定した通りに取捨選択することによりインタフェースを隠蔽し、重要なインタフェース

の秘匿性を確保する。

【0128】

そして、仮想アプリケーションサービス（VAS）841～848のプロセスには、デイスパッチャ144と、ラッピング処理スレッド141と、バージョン管理スレッド142とが動作している。

【0129】

デイスパッチャ144は、アプリケーション130やコントロールサービスからのメッセージ受信を監視し、受信したメッセージに応じてラッピング処理スレッド141、あるいはバージョン管理スレッド142に対して処理要求を行うものである。実施の形態2の複合機800では、デイスパッチャ144は、VAS制御プロセス801から、アプリID、アプリのプロセスIDとともに、バージョン管理要求メッセージまたはラッピング処理要求メッセージを受信している。デイスパッチャ144は、バージョン管理要求メッセージを受信したときには、アプリID、アプリのプロセスIDとともに受信したバージョン管理要求メッセージをバージョン管理スレッド142に送信し、ラッピング処理要求メッセージを受信したときには、アプリID、アプリのプロセスIDとともに受信したラッピング処理要求メッセージをラッピング処理スレッド141に送信するようになっている。

【0130】

バージョン管理スレッド142は、デイスパッチャ144からのバージョン管理要求メッセージを受信すると、実施の形態1におけるVAS140と同様に、バージョン管理に必要な情報を取得してラッピング処理情報ファイル201をハードディスク（HD）200に生成する。

【0131】

ラッピング処理スレッド141は、デイスパッチャ144からのラッピング処理要求メッセージを受信すると、実施の形態1におけるVAS140と同様に、ラッピング処理情報ファイル201を参照して、ラッピング処理が行われる。

【0132】

実施の形態2の複合機800におけるVAS841～848のバージョン管理



スレッド 1 4 2 によって実行されるバージョン管理処理、およびラッピング処理  
スレッド 1 4 1 によって実行されるラッピング処理については、実施の形態 1 の  
複合機 1 0 0 における V A S 1 4 0 の各スレッドによる処理と同様であるので、  
重複説明を省略する。

#### 【 0 1 3 3 】

このように実施の形態 2 にかかる複合機 8 0 0 によれば、実施の形態 1 にかかる  
複合機 1 0 0 と同様に、アプリケーションとコントロールサービスおよびイン  
タフェースとの間に生じたバージョン差の整合性を判断し、サポート可能な範囲  
のバージョン差であればこれを吸収すると共に、アプリケーションとコントロー  
ルサービス間の重要なインタフェースを隠蔽して秘匿性を確保しつつ、特定のイ  
ンタフェースは開示して新規アプリケーションの開発効率を向上させることがで  
きる。

#### 【 0 1 3 4 】

また、実施の形態 2 にかかる複合機 8 0 0 では、V A S 8 4 1 ~ 8 4 8 は起動  
されるアプリケーション 1 3 0 ごとに別個に起動されるので、複数のアプリケー  
ション 1 3 0 の起動判断処理を各アプリケーション 1 3 0 に対応する V A S 8 4  
1 ~ 8 4 8 で並列に実行することができ、アプリケーションの起動判断処理が効  
率的に行えると共に、上記したバージョン管理やラッピング処理がアプリごとに  
行えるので、新規アプリの追加や交換時にも効率良く行うことができる。

#### 【 0 1 3 5 】

なお、実施の形態 1 および 2 にかかる複合機 1 0 0、8 0 0 では、ラッピング  
処理情報ファイル 2 0 1 をハードディスク 2 0 0 に格納し、バージョン管理テー  
ブル 2 1 1 を R A M 2 1 0 に格納して実施したが、かかる格納場所や格納形態は  
一例であり、他の格納場所や格納形態を採用することも勿論可能である。

#### 【 0 1 3 6 】

##### 【発明の効果】

以上説明したように、請求項 1 にかかる発明によれば、コントロールサービス  
のいずれかのコンポーネントがバージョンアップしたとき、ラッピング処理手段  
によってバージョン差の生じたアプリケーションからの関数呼び出しに対して整

合性をとるようにしたので、バージョン差が吸収され、バージョン不整合による不具合発生を防止することができる。

## 【 0 1 3 7 】

また、請求項 2 にかかる発明によれば、アプリケーションプログラムインタフェースがバージョンアップしたとき、ラッピング処理手段によってバージョン差の生じたアプリケーションからの関数呼び出しに対して整合性をとるようにしたので、バージョン差が吸収され、バージョン不整合による不具合発生を防止することができる。

## 【 0 1 3 8 】

また、請求項 3 にかかる発明によれば、ラッピング処理手段によってコントロールサービスからアプリケーションに対するメッセージを取捨選択するので、サードベンダなどの第三者に対してはコントロールサービスとの間のインタフェースを隠蔽し、重要なインタフェースの秘匿性を確保して、第三者からコントロールサービスに直接アクセスされるのを回避することにより、画像形成装置におけるセキュリティ面が確保され、障害発生を未然に防止することができる。

## 【 0 1 3 9 】

また、請求項 4 にかかる発明によれば、関数または関数の引数が追加されてバージョンアップした場合、ラッピング処理手段はアプリケーションに対して提供する関数または関数の引数を変更しなければ、追加された機能は使えないが既存の機能はそのまま使えるので、追加機能は無視すればアプリケーションを再コンパイルすることなくバージョン差を吸収することができる。

## 【 0 1 4 0 】

また、請求項 5 にかかる発明によれば、関数または関数の引数が分割されてバージョンアップした場合、対応する関数や関数の引数が増えるので、そのままでは関数呼び出しを行うことができず、ラッピング処理手段によって増加した関数や関数の引数に対応するダミーの関数または関数の引数を補填することによって整合性をとることができ、バージョン差を吸収することができる。

## 【 0 1 4 1 】

また、請求項 6 にかかる発明によれば、特定のメッセージをアプリケーション

に対して通知しないように、ラッピング処理手段に対して予め設定することが可能であるので、重要なインタフェースの秘匿性を確保することができる。

## 【0142】

また、請求項7にかかる発明によれば、ラッピング処理手段は仮想アプリケーションサービスに含まれており、この仮想アプリケーションサービスは、コントロールサービスをサーバとしたクライアントプロセスとして動作し、かつアプリケーションをクライアントとしたサーバプロセスとして動作するので、バージョン差が生じた場合にバージョン差を吸収して整合性をとることができ、また、コントロールサービスからアプリケーションに対するメッセージを選択することによって、インタフェースを隠蔽して重要なインタフェースの秘匿性を確保しつつ、新規アプリケーションのプログラム開発を効率的に行うことができる。

## 【0143】

また、請求項8にかかる発明によれば、ラッピング処理手段は、仮想アプリケーションサービスのプロセス内部にスレッドとして生成され、実行されるので、仮想アプリケーションサービスのプロセス内部で軽快に動作させることができると共に、プロセス内部に複数個のスレッドを生成した場合であっても、それらを並行処理することができる。

## 【0144】

また、請求項9にかかる発明によれば、アプリケーションの実行時に、共通の関数を利用して他のプログラムと結合させることにより実行形式のプログラムを作成する動的リンク手段を備えているので、アプリケーションとコントロールサービスあるいはインタフェース間のバージョン差の吸収力が増大し、整合性が一層とれるようになると共に、インタフェースを隠蔽して重要なインタフェースの秘匿性をより強固に確保することができる。

## 【0145】

また、請求項10にかかる発明によれば、アプリケーションとコントロールサービスのいずれかのコンポーネントとの間でバージョン差を生じた場合に、そのバージョン差がサポート可能な範囲か否かを判定し、その判定結果をアプリケーションに通知するようにしたので、例えばアプリケーションの登録時などにバー

ジョンの整合性を判定すれば、アプリケーションの実行前にバージョンの適合性の有無が判断できることから、障害の発生を未然に防止することができる。

## 【 0 1 4 6 】

また、請求項 1 1 にかかる発明によれば、アプリケーションとアプリケーションプログラムインタフェースとの間でバージョン差が生じた場合に、そのバージョン差がサポート可能な範囲か否かを判定し、その判定結果をアプリケーションに通知するようにしたので、例えばアプリケーションの登録時などにバージョンの整合性を判定すれば、アプリケーションの実行前にバージョンの適合性の有無が判断できることから、障害の発生を未然に防止することができる。

## 【 0 1 4 7 】

また、請求項 1 2 にかかる発明によれば、バージョン管理手段がサポート可能な範囲のバージョンか否かをアプリケーションのバージョン番号に基づいて判定するようにしたので、容易かつ正確に判定することができる。

## 【 0 1 4 8 】

また、請求項 1 3 にかかる発明によれば、バージョン管理手段がサポート可能な範囲のバージョン番号が登録されたバージョン管理テーブルを備えていて、アプリケーションのバージョン番号をバージョン管理テーブルと比較して判定するようにしたので、サポート可能なアプリケーションか否かを容易かつ正確に判定することができる。

## 【 0 1 4 9 】

また、請求項 1 4 にかかる発明によれば、バージョン管理手段によりアプリケーションの使用する関数を全種類抽出し、その抽出した各関数と、コントロールサービスとの間にバージョン差が生じた場合に、サポート可能な範囲のバージョン差か否かを判定して、その判定結果をアプリケーションに通知するようにしたので、アプリケーションが使用していない関数がバージョンアップされてもバージョン管理手段によるバージョンチェックに影響を与えないことから、サポート可能な範囲を広げることができる。また、サポートが不可能なバージョン差があると判定されても、その原因となる関数を容易に特定することができる。

## 【 0 1 5 0 】

また、請求項 1 5 にかかる発明によれば、バージョン管理手段によりアプリケーションの使用する関数を全種類抽出し、その抽出した各関数と、アプリケーションプログラムインタフェースとのバージョンとの間にバージョン差が生じた場合に、サポート可能な範囲のバージョン差か否かを判定して、その判定結果をアプリケーションに通知するようにしたので、アプリケーションが使用していない関数がバージョンアップされてもバージョン管理手段によるバージョンチェックに影響を与えないことから、サポート可能な範囲を広げることができる。また、サポートが不可能なバージョン差があると判定されても、その原因となる関数を容易に特定することができる。

## 【 0 1 5 1 】

また、請求項 1 6 にかかる発明によれば、バージョン管理手段がサポート可能な範囲のバージョンか否かを判定するにあたって、アプリケーションが使用する全ての関数の番号とバージョン番号に基づいて行われるので、単一の番号比較だけでできることから管理が容易となり、バージョンチェック処理の負荷が小さくなるという利点がある。

## 【 0 1 5 2 】

また、請求項 1 7 にかかる発明によれば、バージョン管理手段は、バージョン管理テーブルに登録されているサポート可能な範囲の全ての関数番号とバージョン番号と、アプリケーションが使用する全ての関数番号とバージョン番号とを比較して判定するようにしたので、バージョン差の生じたアプリケーションがサポート可能か否かを容易かつ正確に判定することができる。

## 【 0 1 5 3 】

また、請求項 1 8 にかかる発明によれば、バージョン管理手段は、コントロールサービスをサーバとしたクライアントプロセスとして動作し、かつアプリケーションをクライアントとしたサーバプロセスとして動作する仮想アプリケーションサービスに含まれているので、バージョン差が生じた場合にバージョン差を吸収して整合性をとることができ、また、コントロールサービスからアプリケーションに対するメッセージを選択することによって、インタフェースを隠蔽して重要なインタフェースの秘匿性を確保しつつ、新規アプリケーションのプログラム

開発を効率的に行うことができる。

【0154】

また、請求項19にかかる発明によれば、バージョン管理手段は、仮想アプリケーションサービスのプロセス内部にスレッドとして生成され、実行されるので、仮想アプリケーションサービスのプロセス内部で軽快に動作させることができると共に、プロセス内部に複数個のスレッドを生成した場合であっても、それらを並行処理することができる。

【0155】

また、請求項20にかかる発明によれば、コントロールサービスのいずれかのコンポーネントがバージョンアップしたとき、ラッピング処理ステップによってバージョン差の生じたアプリケーションからの関数呼び出しに対して整合性をとるので、バージョン差が吸収され、バージョン不整合による不具合発生を防止することができる。

【0156】

また、請求項21にかかる発明によれば、アプリケーションプログラムインタフェースがバージョンアップしたとき、ラッピング処理ステップによってバージョン差の生じたアプリケーションからの関数呼び出しに対して整合性をとるようにしたので、バージョン差が吸収され、バージョン不整合による不具合発生を防止することができる。

【0157】

また、請求項22にかかる発明によれば、ラッピング処理ステップによってコントロールサービスからアプリケーションに対するメッセージを取捨選択するので、サードベンダなどの第三者に対してコントロールサービスとの間のインタフェースを隠蔽することで、重要なインタフェースの秘匿性を確保し、第三者からコントロールサービスに直接アクセスされるのを回避して、画像形成装置におけるセキュリティ面を確保することにより、障害の発生が未然に防止できる。

【0158】

また、請求項23にかかる発明によれば、ラッピング処理ステップにおいて、関数または関数の引数が追加されてバージョンアップした場合、追加された関数

または関数の引数に対応する機能が無視して整合性をとるようにしたので、アプリケーションからの関数呼び出しに使われる関数または関数の引数をそのまま使うことができ、アプリケーションを再コンパイルすることなくバージョン差を吸収することができる。

## 【 0 1 5 9 】

また、請求項 2 4 にかかる発明によれば、ラッピング処理ステップにおいて、関数または関数の引数が分割されてバージョンアップした場合、分割された関数または関数の引数に応じてダミーの関数または関数の引数を補填して整合性をとるようにしたので、関数の対応関係を維持することが可能となり、アプリケーションを再コンパイルすることなくバージョン差を吸収することができる。

## 【 0 1 6 0 】

また、請求項 2 5 にかかる発明によれば、ラッピング処理ステップにおいて、アプリケーションに対して通知しない特定のメッセージが予め設定可能なので、サードベンダなどに公開するインタフェースの基準を任意に決定することができ、重要なインタフェースの秘匿性を確保することができる。

## 【 0 1 6 1 】

また、請求項 2 6 にかかる発明によれば、アプリケーションとコントロールサービスのいずれかのコンポーネントとの間でバージョン差が生じた場合に、バージョン管理ステップによりバージョン差がサポート可能な範囲か否かを判定し、その判定結果をアプリケーションに通知するようにしたので、例えばアプリケーションの登録時などにバージョンの整合性を判定すれば、アプリケーションの実行前にバージョンの適合性の有無が判断でき、障害の発生を未然に防止することができる。

## 【 0 1 6 2 】

また、請求項 2 7 にかかる発明によれば、アプリケーションとアプリケーションプログラムインタフェースとの間でバージョン差が生じた場合に、バージョン管理ステップによりバージョン差がサポート可能な範囲か否かを判定し、その判定結果をアプリケーションに通知するようにしたので、例えばアプリケーションの登録時などにバージョンの整合性を判定すれば、アプリケーションの実行前に

バージョンの適合性の有無が判断でき、障害の発生を未然に防止することができる。

## 【 0 1 6 3 】

また、請求項 2 8 にかかる発明によれば、バージョン管理ステップにおいて、サポート可能な範囲のバージョンか否かをアプリケーションのバージョン番号に基づいて判定するようにしたので、容易かつ正確に判定することができる。

## 【 0 1 6 4 】

また、請求項 2 9 にかかる発明によれば、バージョン管理ステップにおいて、サポート可能な範囲のバージョン番号が登録されたバージョン管理テーブルを用い、登録されるアプリケーションのバージョン番号をバージョン管理テーブルと比較して判定するようにしたので、サポート可能なアプリケーションか否かを容易かつ正確に判定することができる。

## 【 0 1 6 5 】

また、請求項 3 0 にかかる発明によれば、バージョン管理ステップによりアプリケーションの使用する関数を全種類抽出し、その抽出した各関数と、コントロールサービスとの間にバージョン差が生じた場合に、サポート可能な範囲のバージョン差か否かを判定して、その判定結果をアプリケーションに通知するようにしたので、アプリケーションが使用していない関数がバージョンアップされてもバージョン管理ステップによるバージョンチェックに影響を与えないことから、サポート可能な範囲を広げることができる。また、サポートが不可能なバージョン差があると判定されても、その原因となる関数を容易に特定することができる。

## 【 0 1 6 6 】

また、請求項 3 1 にかかる発明によれば、バージョン管理ステップによりアプリケーションの使用する関数を全種類抽出し、その抽出した各関数と、アプリケーションプログラムインタフェースとのバージョンとの間にバージョン差が生じた場合に、サポート可能な範囲のバージョン差か否かを判定して、その判定結果をアプリケーションに通知するようにしたので、アプリケーションが使用していない関数がバージョンアップされてもバージョン管理ステップによるバージョン



チェックに影響を与えないことから、サポート可能な範囲を広げることができる。また、サポートが不可能なバージョン差があると判定されても、その原因となる関数を容易に特定することができる。

## 【 0 1 6 7 】

また、請求項 3 2 にかかる発明によれば、バージョン管理ステップによりサポート可能な範囲のバージョンか否かを判定するにあたって、アプリケーションが使用する全ての関数の番号とバージョン番号に基づいて行われるので、単一の番号比較だけでできることから管理が容易となり、バージョンチェック処理の負荷が小さくなるという利点がある。

## 【 0 1 6 8 】

また、請求項 3 3 にかかる発明によれば、バージョン管理ステップでは、バージョン管理テーブルに登録されているサポート可能な範囲の全ての関数番号とバージョン番号と、アプリケーションが使用する全ての関数番号とバージョン番号とを比較して判定するようにしたので、バージョン差の生じたアプリケーションがサポート可能か否かを容易かつ正確に判定することができる。

## 【図面の簡単な説明】

## 【図 1】

実施の形態 1 にかかる複合機の構成を示すブロック図である。

## 【図 2】

実施の形態 1 にかかる複合機の V A S の構成と、V A S と各アプリ、コントロールサービス層および汎用 O S との関係を示すブロック図である。

## 【図 3】

H D に格納されるラッピング処理情報ファイルの内容例を示す説明図である。

## 【図 4】

R A M に格納される一括管理用のバージョン管理テーブルの内容例を示す説明図である。

## 【図 5】

R A M に格納される関数単位管理用のバージョン管理テーブルの内容例を示す説明図である。

【図 6】

V A S の制御スレッドによりバージョン差を吸収するためのラッピング処理を行う手順を示すフローチャートである。

【図 7】

V A S の制御スレッドによりコントロールサービスの特定のインタフェースを隠蔽するためのラッピング処理を行う手順を示すフローチャートである。

【図 8】

V A S の制御スレッドによりアプリ実行の初期段階でバージョン不整合があるか否かを判定するバージョン管理のための手順を示すフローチャートである。

【図 9】

V A S の制御スレッドによりアプリが実際に使用している関数を全種類抽出し関数単位でバージョンの不整合があるか否かを判定するバージョン管理の手順を示すフローチャートである。

【図 1 0】

実施の形態 2 にかかる複合機の構成を示すブロック図である。

【図 1 1】

実施の形態 2 にかかる複合機の V A S の構成と、V A S と各アプリ、コントロールサービス層および汎用 O S との関係を示すブロック図である。

【符号の説明】

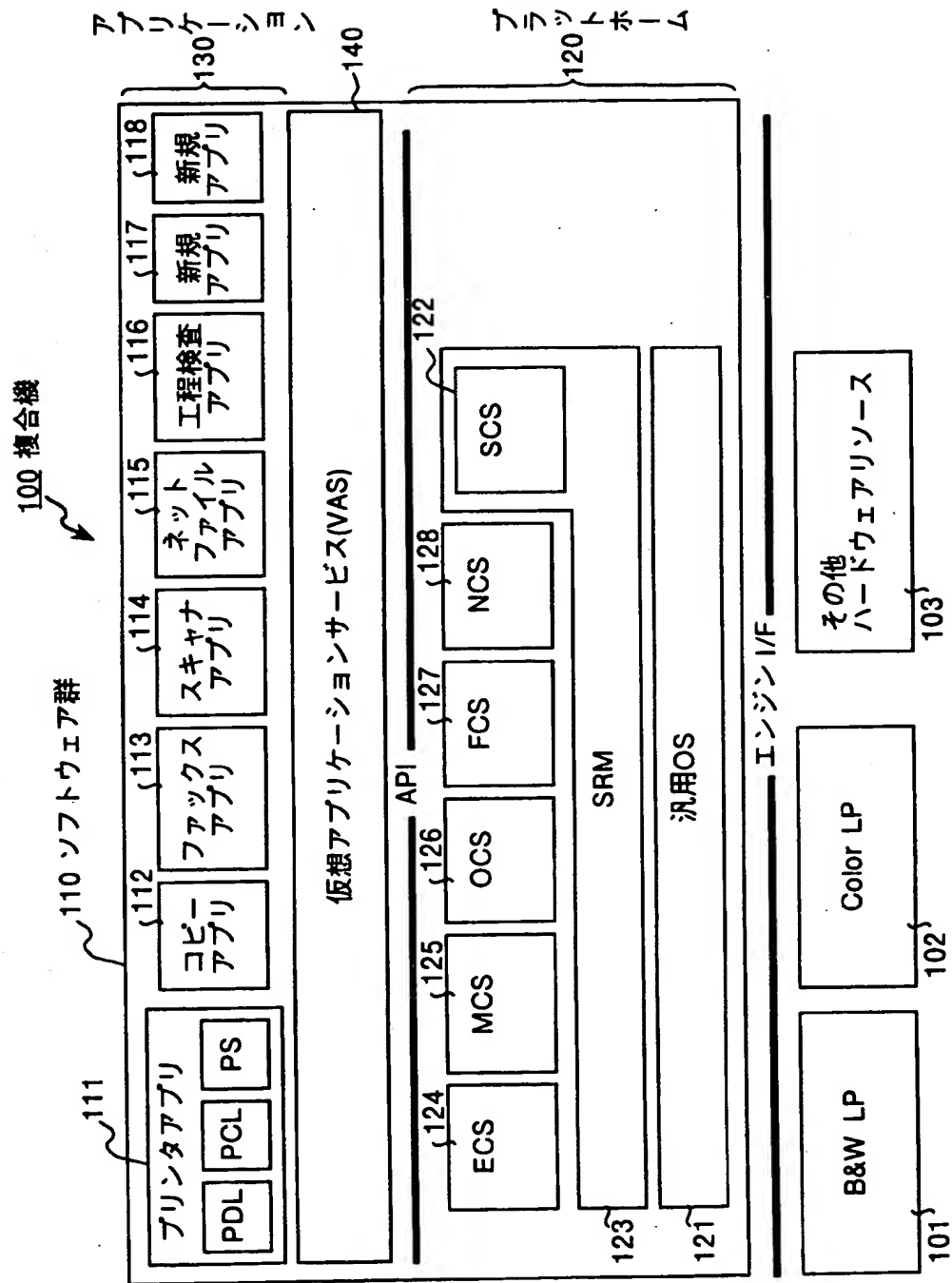
- 1 0 0 複合機
- 1 0 1 白黒レーザプリンタ
- 1 0 2 カラーレーザプリンタ
- 1 0 3 ハードウェアリソース
- 1 1 0 ソフトウェア群
- 1 1 1 プリンタアプリ
- 1 1 2 コピーアプリ
- 1 1 3 ファックスアプリ
- 1 1 4 スキャナアプリ
- 1 1 5 ネットファイルアプリ

- 1 1 6 工程検査アプリ
- 1 1 7、1 1 8 新規アプリ
- 1 2 0 プラットホーム
- 1 2 1 汎用OS
- 1 2 2 SCS
- 1 2 3 SRM
- 1 2 4 ECS
- 1 2 5 MCS
- 1 2 6 OCS
- 1 2 7 FCS
- 1 2 8 NCS
- 1 3 0 アプリケーション
- 1 4 0、8 4 1～8 4 8 仮想アプリケーションサービス (VAS)
- 1 4 1 ラッピング処理スレッド
- 1 4 2 バージョン管理スレッド
- 1 4 3 制御スレッド
- 1 4 4 ディスパッチャ
- 1 5 0 コントロールサービス層
- 2 0 0 ハードディスク (HD)
- 2 0 1 ラッピング処理情報ファイル
- 2 1 0 RAM
- 2 1 1 バージョン管理テーブル
- 8 0 0 複合機
- 8 0 1 VAS制御プロセス

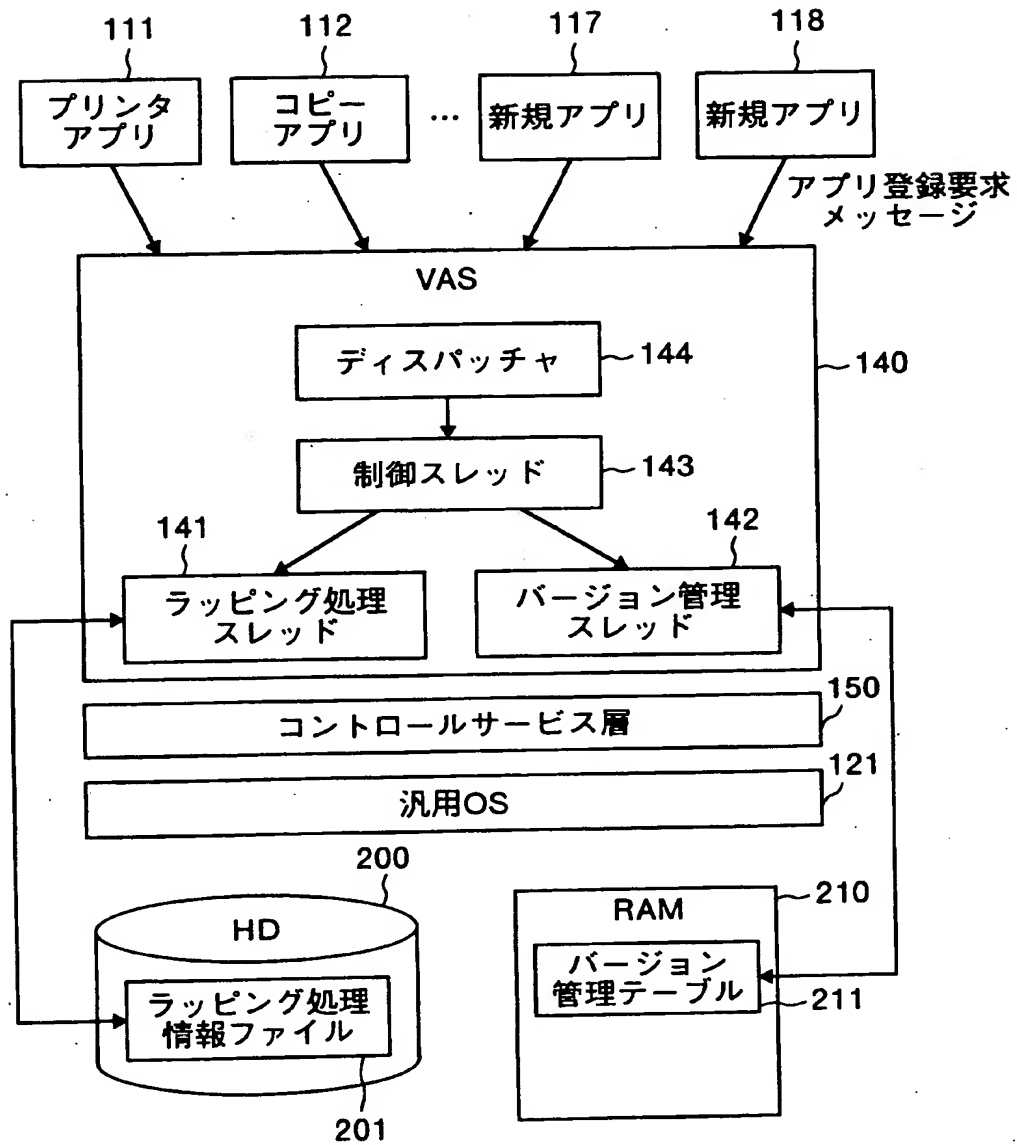
【書類名】

図面

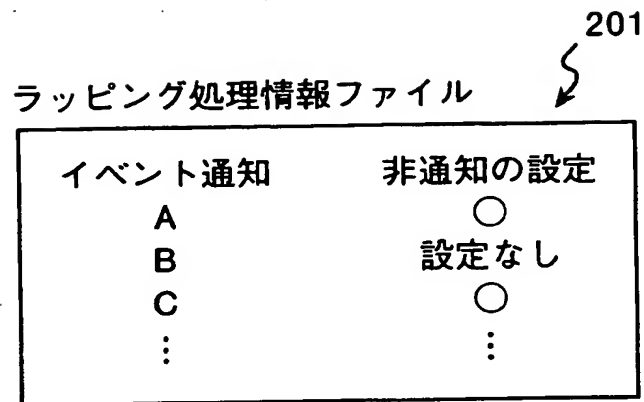
【図 1】



【図 2】



【図 3】



【図 4】

211

バージョン管理テーブル（一括管理用）

アプリID	バージョン番号	サポート範囲
101	1.6	1.0～1.6
102	1.4	1.0～1.4
103	1.8	1.1～1.8
⋮	⋮	⋮

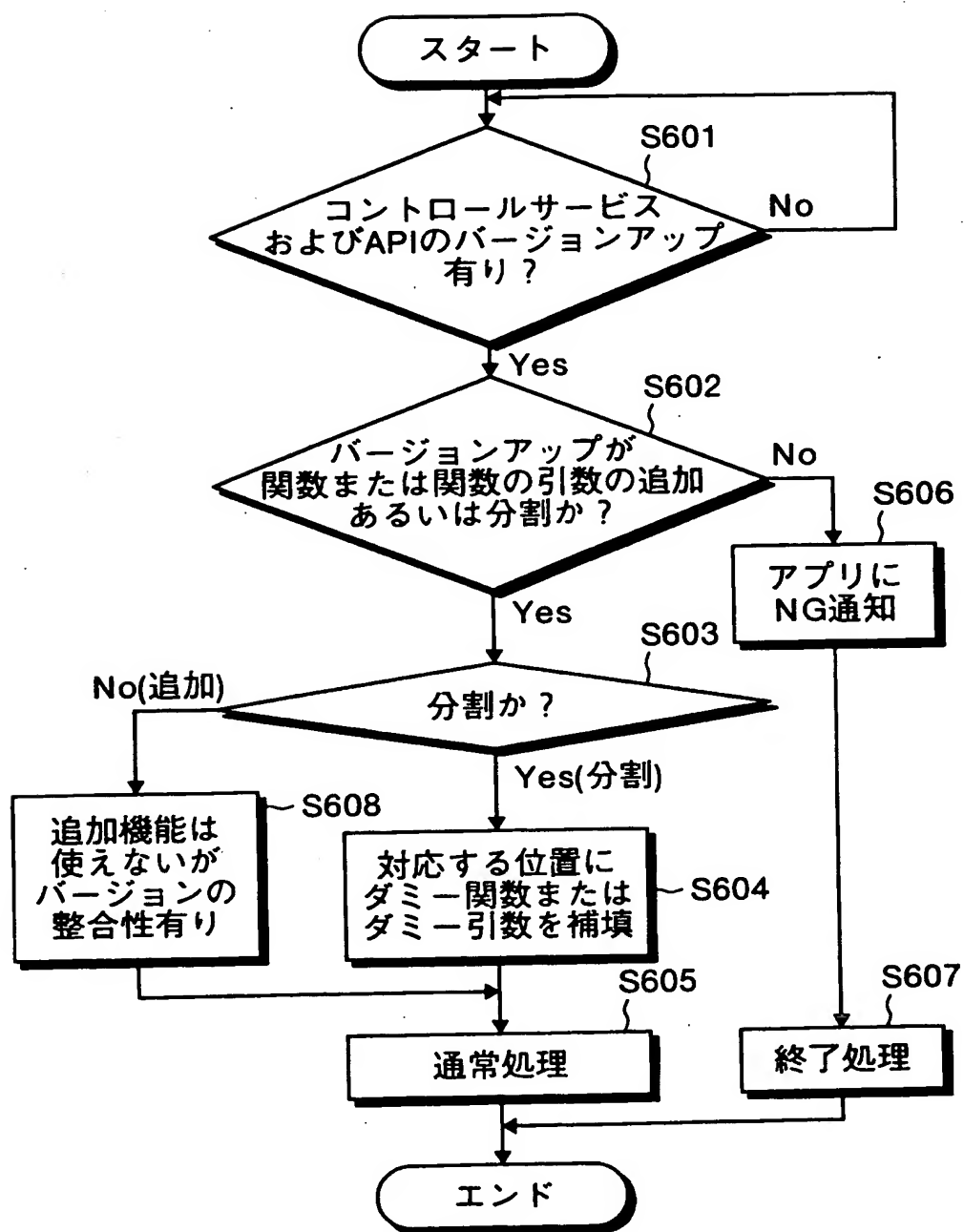
【図 5】

211  
↙

バージョン管理テーブル（関数単位管理用）

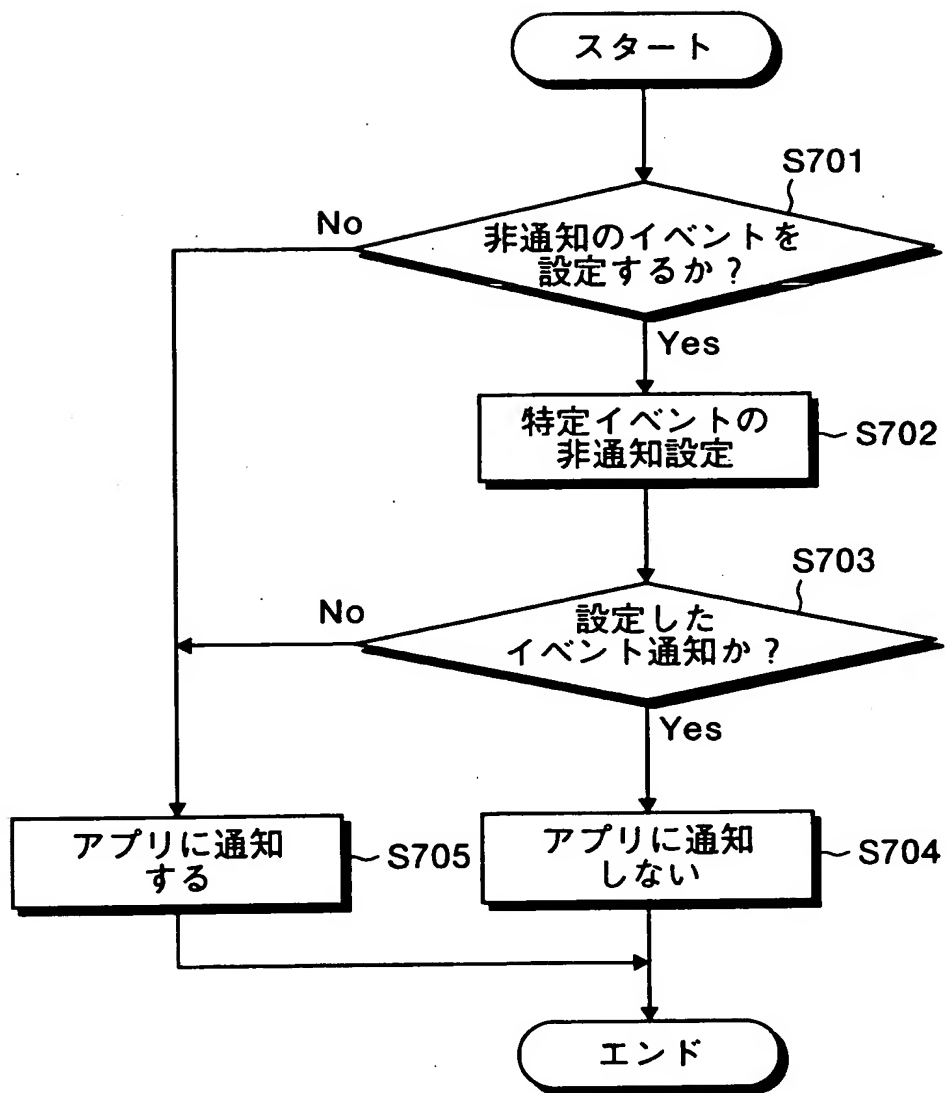
関数番号	バージョン番号	サポート範囲	使用の有無
1	1.4	1.0~1.4	×
2	1.2	1.0~1.2	○
3	2.3	2.0~2.3	○
4	1.1	1.0~1.1	×
⋮	⋮	⋮	⋮
n	⋮	⋮	⋮

【図 6】

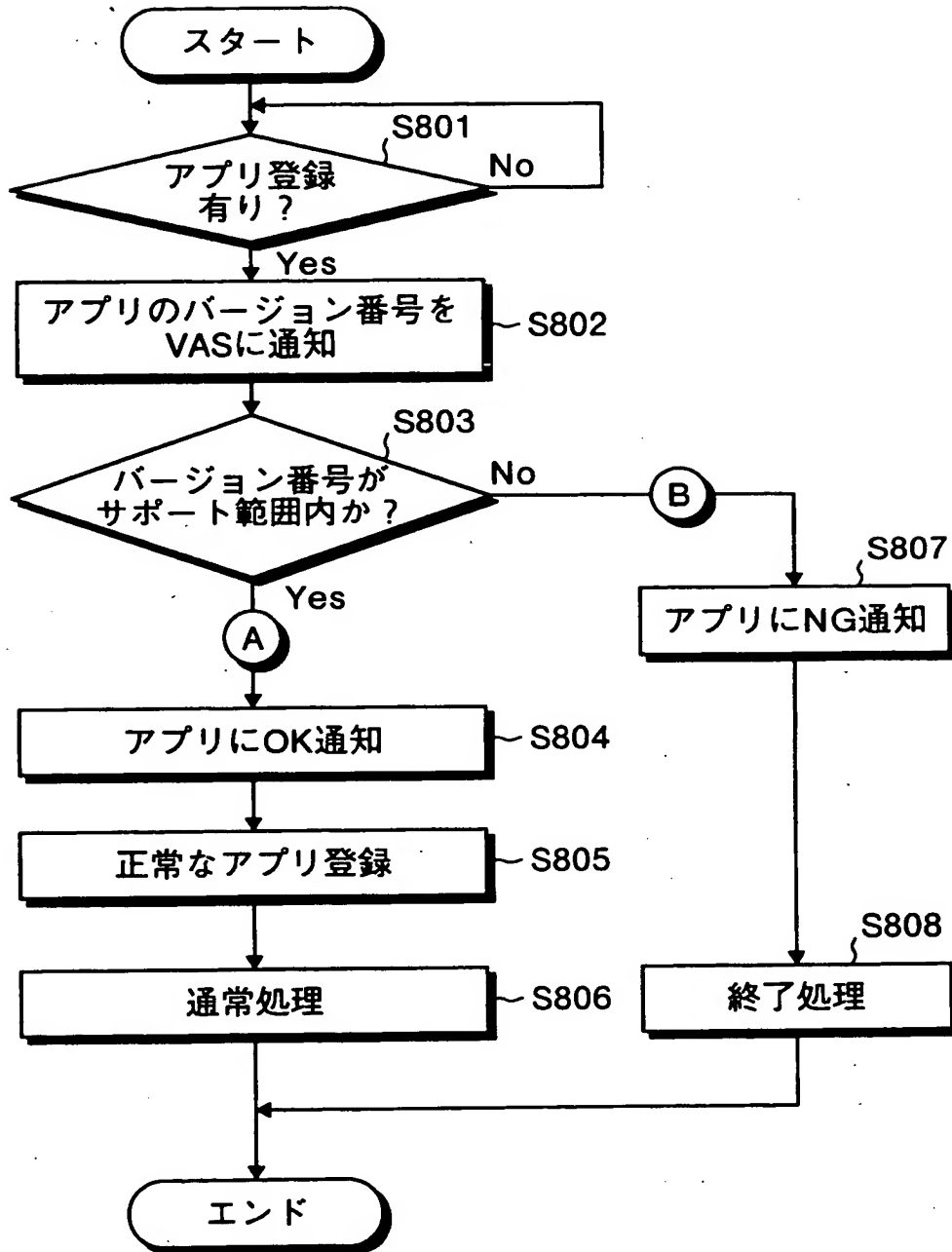




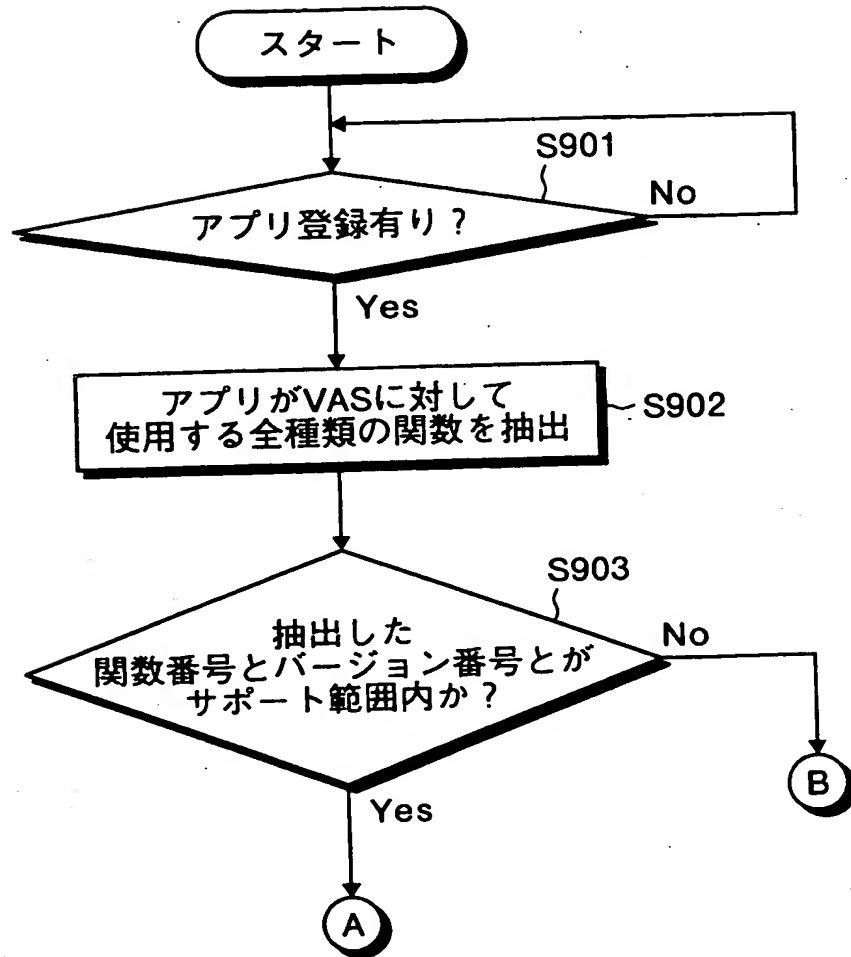
【図 7】



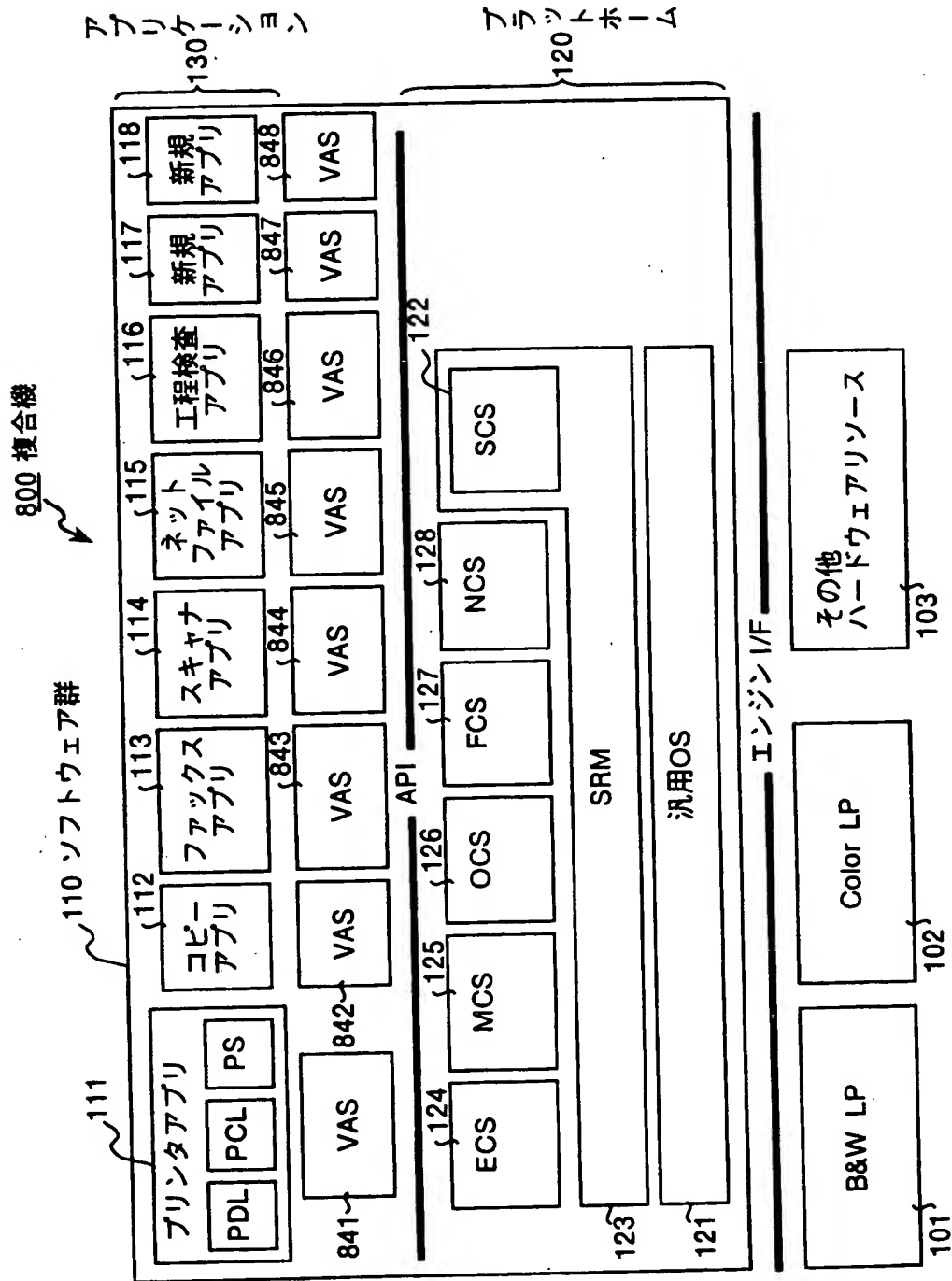
【図 8】



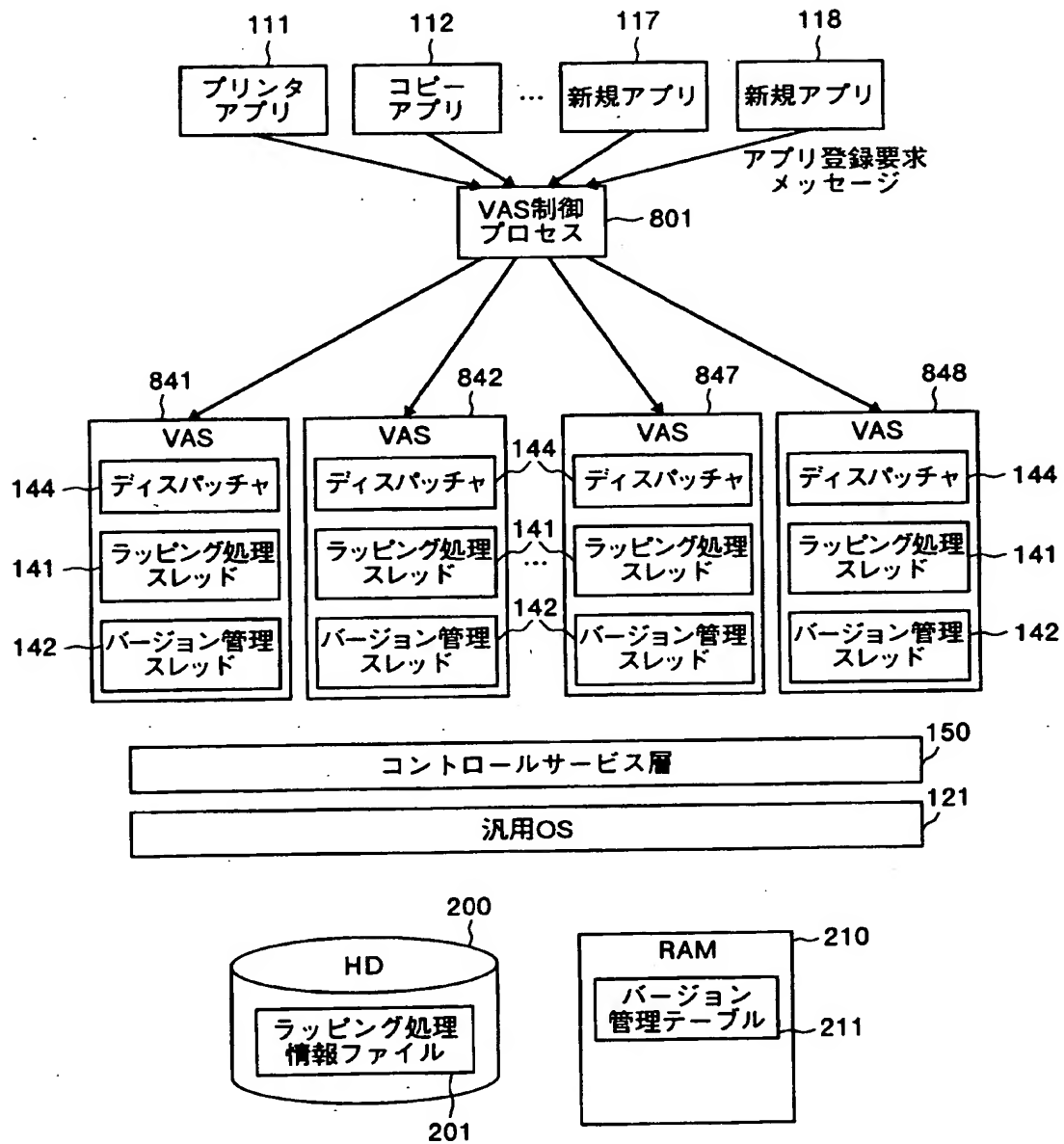
【図 9】



【図10】



【図 11】



【書類名】 要約書

【要約】

【課題】 バージョン差を吸収して、重要なインタフェースの秘匿性を確保しつつ、バージョンチェックを容易かつ確実に行えるようにする。

【解決手段】 VAS 140 のラッピング処理スレッド 141 により、バージョン差を吸収して整合性を保持すると共に、コントロールサービス層 150 からアプリに対するメッセージ通知を取捨選択することにより、重要なインタフェースを第三者に対して隠蔽することが可能となる。また、VAS 140 のバージョン管理スレッド 142 は、アプリのバージョン番号、あるいは、アプリが VAS 140 に対して実際に使用している関数番号とそのバージョン番号とをバージョン管理テーブル 211 に対して個々に比較することにより、サポート可能な範囲のバージョン違いか否かを容易かつ確実に判定することができる。

【選択図】 図 2

出 願 人 履 歴 情 報

識別番号 [000006747]

1. 変更年月日 2002年 5月17日  
[変更理由] 住所変更  
住 所 東京都大田区中馬込1丁目3番6号  
氏 名 株式会社リコー